NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ANALYZING THREADS AND PROCESSES IN WINDOWS CE

by

Titus R. Burns

September 2001

Thesis Advisor: Cynthia E. Irvine
Second Reader: Paul Clark

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
30 Sep 2001	N/A	-

Title and Subtitle Analyzing Threads and Processes in Windows CE	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Titus R. Burns	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 102	

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Analyzing Threads and Processes in Windows CE			**5. FUNDING NUMBERS**
6. AUTHOR(S) Titus R. Burns			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

Windows CE 3.0, also known as Pocket PC for palm-sized devices, is becoming increasingly popular among professionals and corporate enterprises. It is estimated that by 2004 Windows CE will have a share of 40% of the marketplace for palm-sized devices. The documented vulnerabilities against a major competitor of WinCE, Palm, and the proliferation of palm-sized devices highlight the need for security for these small-scale systems. This thesis is part of a larger project to enhance the security in WinCE.

This thesis analyzed the threads and processes in WinCE, and discusses authentication, public key infrastructure (PKI) and future technologies as each relates to WinCE. The research discovered that *Talisker*, the next generation of WinCE, supports Kerberos an authentication protocol, and it also supports PKI (a key management system) components. Results of this thesis show that security can be enhanced in WinCE without requiring a change to its code base.

14. SUBJECT TERMS Operating System, Personal Digital Assistant, Security, Threads, Processes, Authentication, Windows CE, Public Key Infrastructure			**15. NUMBER OF PAGES** 102
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ANALYZING THREADS AND PROCESSES IN WINDOWS CE

Titus R. Burns
Captain, United States Marine Corps
B.S., Prairie View A&M University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: Titus R. Burns
Titus R. Burns

Approved by: Cynthia E. Irvine
Cynthia E. Irvine, Thesis Advisor

Paul C. Clark
Paul Clark, Second Reader

Chris Eagle
Chris Eagle, Chairman
Information Systems Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Windows CE 3.0, also known as Pocket PC for palm-sized devices, is becoming increasingly popular among professionals and corporate enterprises. It is estimated that by 2004 Windows CE will have a share of 40% of the marketplace for palm-sized devices. The documented vulnerabilities against a major competitor of WinCE, Palm, and the proliferation of palm-sized devices highlight the need for security for these small-scale systems. This thesis is part of a larger project to enhance the security in WinCE.

This thesis analyzed the threads and processes in WinCE, and discussed authentication, public key infrastructure (PKI) and future technologies as each relates to WinCE. The research discovered that *Talisker*, the next generation of WinCE, supports Kerberos an authentication protocol, and it also supports PKI (a key management system) components. Results of this thesis show that security can be enhanced in WinCE without requiring a change to its code base.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author wants to thank his LORD and SAVIOR, JESUS CHRIST, for his mercy, kindness, and love, to whom all honor, praise and glory belongs forever.

The author wants to thank his wife, Renee Burns, for her love and support.

The author wants to thank Professor Cynthia E. Irvine for her patience, guidance and support during this work.

The author wants to thank Mr. Paul Clark for his guidance and support during this work

The author wants to acknowledge the financial support of the Microsoft Corporation during this work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. PERSONAL DIGITAL ASSISTANT

A. INTRODUCTION

Familiarity and functionality are factors that influence many consumers' purchasing decisions with regard to computers and software. An often-overlooked factor is the information assurance (IA) capabilities of these. These IA factors can assist in creating, for individuals familiar with information assurance issues, a perceived security picture of a system or device. This thesis will discuss the Windows CE (WinCE) operating system, known as the Pocket PC operating system (OS).

Pocket PC is the third generation of the WinCE operating system. Pocket PC executes on Compaq's iPAQ, HP's Jornada, Casio, and Symbol [OWS3j]. *Merlin* is the next operating system release scheduled for Pocket PCs; Merlin is based on WinCE 3.0. [NGS1b] Personal Digital Assistants (PDAs) that execute WinCE offer the Windows familiar look and feel, which may account for the device's popularity among professionals [OWS3l]. Pocket PCs offer many familiar Windows applications, such as: Word, Excel, File Explorer, etc. For individuals who are used to the Windows environment, Pocket PC will appear to be a miniature version of their desktop or laptop.

Windows CE may also appeal to people who like to stay current with new technology and maximum functionality. Windows CE is the only operating system executing on a PDA that will integrate with Microsoft's .NET initiative, which will be discussed in Chapter V [MWS2l]. In comparison with the Palm operating system (the leading OS in the palm-size device market [OWS3l]), WinCE can multitask, provide a robust multimedia capability and supports software emulators for the Nintendo Entertainment System (NES) and Game Boy, Commodore 64, Sega, and others. Windows CE's handwriting recognition system has enhanced handwriting recognition accuracy on the PDA. [OWS3j] This Windows CE functionality may also help account for its popularity in the professional and corporate marketplace.

No matter how popular a device, operating system, or technology in general, becomes in the marketplace, for the security conscious, it is the information assurance capabilities of the device that may be most appealing. For many, information assurance may not mean anything, but when IA is equated to privacy and protection of information, its necessity becomes clear [OWS3m]. There have been numerous exploits against PDAs, specifically the Palm OS [KIN01]. Before discussing some of the exploits, it is important to note that in this work, the assumption is made that these exploits have been targeted against the Palm OS because it is the most common PDA operating system. However, a lack of exploits should not lead to a false sense of security with regard to WinCE.

B. PDA VULNERABILITIES

In August of 2000, the first Palm Pilot Trojan Horse, which attacks Palm operating systems, was identified. The importance of the event was not the Trojan Horse itself, but the possibilities and realization that malicious code could be written that would be harmful to Palms. The implication of this is serious. It now means that the move to wireless and developing Internet appliances that interact with everything from cars to refrigerators are susceptible to viruses and Trojan Horses. [OWS3m]

Personal Digital Assistants software vulnerabilities (exploits) are not only limited to the PDA's. In February 2001, an exploit against passwords on the Palm Desktop Version 4 was published [NGS1a]. This exploit is mentioned so that the reader can understand that information assurance must provide protection of the information on the local system (PDA), in the synchronization protocols, and at the desktop. Even if PDA operating systems are enhanced to provide security, the host system that it is partnered with for synchronization of databases must provide adequate security to ensure that the information is protected. However, the scope of this thesis is on the WinCE operating system rather than the desktop services provided for the PDA.

C. PDA SECURITY

Developers of the Palm OS and Pocket PC OS have installed a password protection option [OWS3d]. However, security must be implemented at every level of the device, i.e. for the applications and the operating system, if the device is ever to be

recognized as a secure system. The goal of this work is to enhance the security of the WinCE operating system. This work is part of an incremental approach to enhancing WinCE security to make it a viable option for use in various environments within the military as well as the corporate workplace. Chapters II through V describe this work. Each chapter is summarized below.

1. Chapter II (Windows NT/UNIX)

Chapter II discusses the two most popular desktop operating systems in use today, Windows NT and UNIX (LINUX is a flavor of UNIX). This chapter provides the reader with an understanding of the threads and processes in WINNT and UNIX. It also provides an overview of the WINNT authentication mechanism. Windows NT, which can be configured to achieve a satisfactory security posture (see Chapter II, Section F), provides a useful example [SOL98]. This background is provided to give the reader an understanding of the thread and process model (WINNT) used for the WinCE thread and process data structures.

2. Chapter III (WinCE)

Chapter III discusses Windows CE processes and threads. This chapter details the creation and termination of WinCE processes and threads. In particular, the reader should take note of security attributes inherit to both the process and thread data structures. *Talisker*, the next generation of the WinCE operating system, is discussed along with some of its improvements. Talisker's improvements include new modules that can be used to enhance security of an operating system image on a device.

3. Chapter IV (Authentication)

This chapter introduces authentication. Authentication is defined as a way to confirm the identity of the client or server in the communication process. Several authentication protocols are discussed in this chapter including: Kerberos and the Department of Defense's Public Key Infrastructure (DoD PKI). An overview of both protocols is provided.

4. Chapter V (Emerging Technology)

This chapter discusses the research and development of new technology. Specifically discussed are processor research and development and the impact it may have on the future of technology. The primary focus of this chapter is on Microsoft's .NET initiative. The objective is to help the reader understand the significance of WinCE as it relates the .NET initiative and palm-sized devices.

5. Chapter VI (Conclusion and Recommendations)

This chapter concludes the thesis. This chapter also analyzes the information presented in the first five chapters and lists recommendations for further research. This chapter summarizes the thesis and lists follow-on questions that may require additional research.

D. CONCLUSION

This thesis provides the reader with a basic understanding of why security is important in operating systems. It also reveals similarities between the WINNT and WinCE process and thread data structures and attributes. This background information gives context to providing security enhancements, such as authentication, to WinCE. However, before delving into security, a brief overview of operating systems is discussed in the next chapter.

II. OVERVIEW OF OPERATING SYSTEMS, PROCESSES AND THREADS

A. INTRODUCTION

Computers have become an integral part of everyday life for many people. They are pervasive in government, the military, in many industries and in many institutions ranging from educational to religious. More and more people are acquiring personal computers for personal and business use. They continue to enhance areas of our lives as never before – so much so that superficial and real dependencies have developed. Computers are widespread and unavoidable; they are a part of our lives whether we know it or not.

Computers are applied to so many aspects of our lives today that our recent past – before our daily exposure to computers – seems like ancient history. For example, consider how we banked twenty-five years ago. To have some money (cash) for the weekend, the customer had to go to the bank on Friday before the bank closed and interact with a teller to access his or her money. Consider banking today: money is accessed from the nearest automated teller machine (provided there are adequate funds available). But even more recently, consider how five years ago how we kept track of important events that we either needed to attend or plan. We used bulky planners with calendars and memos. Today, many people use Personal Digital Assistants (PDAs).

In the April 2001 issue of PC World magazine [OWS3b], one consumer acknowledges using his PDA for work, leisure, and dating. The article goes on to describe how the consumer uses his IPaq (a palm-sized device made by Compaq running the Windows CE operating system) to take notes during company meetings, to catch up on his email messages while lying in bed, and to plug in his headphones to listen to MP3s at the gym. In the same article another consumer expressed his appreciation for this Handspring Visor Deluxe, a PDA based on the Palm operating system. With more and

more professionals and consumers using PDAs, the Gartner Group estimates sales in the United States over the next four years to hit 28 million, an increase of 300% [OWS3b].

Although PDAs may not equal or exceed the number of desktop computers in the market place for some years to come, the two main electronic organizer operating systems (Palm OS and Windows CE OS) are quickly heading for a showdown [OWS3c]. This fierce competition is resulting in consumers reaping many benefits. Benefits from the Pocket PCs are: bright 240x320 screens, fast StrongARM processors, and familiar Windows-like interfaces; while Palm PDAs are delivering better streamlined operations and longer battery life (Palm has a different battery model than Pocket PC) [OWS3c]. As these two products continue to compete, consumers will soon realize the strengths of both products in one PDA, and analysts estimate this will bring near equilibrium to the marketplace by 2004 with Windows CE taking 40% of the market share while Palm takes 45% [OWS3c]. PDAs have become so popular that users can even purchase insurance for them [OWS3d].

Insurance on a PDA may provide the owner of the PDA with some comfort in situations involving the loss of the PDA due to theft or in some cases even negligence on the part of the owner (loss of PDA in the airport), but it cannot provide the owner with any guarantee of data recovery. With insurance, the owner has a reasonably high probability that the PDA will be replaced; however, the owner has no possibility of recovering or replacing the data, provided the data was not backed-up elsewhere. Also data can be lost without the loss of the PDA. Malicious software, worms, and viruses can corrupt or contaminate data beyond repair. Insurance can do little for lost data. One of the best ways to try to mitigate such threats is to harden or secure the critical data management software and the operating system.

The WinCE-based architecture includes data management components such as: object store (file system), drivers, shells, etc., all of which either manipulate data directly or indirectly, or manipulate objects that in turn manipulate data. WinCE-based applications are layered on top of the architecture. The software components within the architecture actually provide the information necessary to run the application, store the data from the applications, or provide data to the applications, it is important to secure

these objects (components). If the architecture is insecure, but the applications are secure, there is no reliable mechanism beneath the application level to insure the protection of the information; therefore, the system is not secure.

Any components that have access to data or a dependency relationship with another component or application that has access to data must be secured to begin to achieve system security. However, system security does not end with securing the components. The components, data, applications, etc. are managed by the operating system. Thus it is most critical to secure the operating system so that application developers can be protected against certain security threats.

The goal of this chapter is to review the basic definition of an operating system and its function, and to familiarize the reader with the concepts, data structure and life cycle of OS support for processes and threads. Of course, there are many more parts to an OS, but this work will focus on processes, threads, and authentication. Keep in mind that part of our goal is to analyze the processes and threads in the context of the Microsoft CE OS and determine how, if possible, to enhance CE to make it a more self-protecting OS. The process and thread modules encompass the life cycle of the processes and threads from their creation to their termination and their access to critical components and protected domains. Thus the focus of this chapter will be an overview of operating systems, with particular emphasis on processes and threads.

There are four general types of operating systems: real-time OSs (RTOS); single-user, single task OSs; single-user, multi-tasking OSs; and multi-user OSs [OWSa]. A brief description of OSs will be presented. A general comparison of a process's data structures and life cycle within the two most popular (personal) commercial OS platforms on the market today, the Microsoft Windows NT-based OS (proprietary code) and the Linux OS (open source code based on UNIX), will also be presented. Although processes and threads will be reviewed for both the Unix platform and Windows platform, the Windows platform will be the basis for exploration into the Windows CE process and thread modules (Chapter III).

B. OPERATING SYSTEM BACKGROUND

It is crucial to understand the operating system mechanism in order to fully comprehend and appreciate the role of processes and threads. Typically, OSs share a common goal of supporting the following types of system components [SIL94]:

- Process management,
- Main-memory management,
- Secondary-storage management,
- I/O system management,
- File management,
- Protection system,
- Networking, and
- Command-interpreter system

Operating systems are not limited to only supporting or managing the above listed components. However, to provide a basis for what components define an OS, the above list is sufficient for this paper. For a more in-depth understanding of the supported components of an OS, refer to any introductory book on operating systems, for example: [SIL98], [STA98], or [TAN97]. Also, understanding what components define an OS enables us to intuitively appreciate the purpose of an operating system.

An operating system is a mechanism or program that simplifies the management of resources (components) to execute application programs. The OS's role in managing resources is what makes securing it difficult. The OS must be robust and flexible enough to support new components as they are introduced into the market. If security on the OS makes it too inflexible, it may not appeal to enough people to survive its competition.

The OS also acts as an intermediary between the user of a computer and the computer hardware. Stallings [STA98] defines an OS as a mechanism that

> exploits the hardware resources of one or more processors to provide a set of services to system users.

Operating systems vary in purpose and design as our use of technology and resources varies, as illustrated by our migration from multi-user, multi-tasking systems to single-user, multi-tasking systems. We have defined the OS at a high level of abstraction. This level of abstraction is both sufficient and convenient for the purpose of discussing processes and threads.

Processes and threads may vary slightly in meaning depending on the particular OS (platform) implementation; however, their roles remain consistent irrespective of the OS. The following sections will discuss processes and threads in general terms and as they each relate to the Unix and Windows OS platforms. If processes and threads differ in definition or roles between the two systems, those differences will be discussed. This approach will provide a greater understanding of processes and threads from a broad perspective while allowing for a detailed analysis of their roles and meanings within, arguably, the two dominant platforms used today.

C. PROCESSES

Processes and threads are necessary to the OS, in part, because the OS utilizes both as a means of executing programs. Threads will be covered later in Section D of this chapter. This section will focus on processes. The term process has been given many different definitions. Generically, many operating system books will define a process as a program in execution, or an entity that can be assigned to and executed on a processor. One book defines it as the "animated spirit" of a program [STA98]. In the Windows NT OS (WINNT), a process is defined as a set of resources reserved for the threads to execute a program [SOL98]. The Unix OS defines a process as an instance of a program in execution [BAC86].

The myriad of definitions may seem to add ambiguity to the term process, but the definitions are not in conflict with one another. Just the opposite is true; the definitions support and enhance one another. A process is an entity that can be assigned to and executed on a processor. Moreover, a process is also an entity that reserves resources so that it can use those resources to execute a program. And lastly, a process is an instance of a program (this definition will prove to be useful in the discussion of swapping and the

“context of a process”). As can be seen, the role of a process is given in its definition – an entity that reserves system resources for the execution of a program.

A process is not a program (application). A program is a static sequence of instructions (text). Programs and processes are synergistic. The program needs the process in order to execute; the process needs the program in order to have purpose. It is sometimes easy to confuse the two because of their dependency. We further distinguish the two by defining a program as a passive entity and a process as an active entity. A process generally includes the current activity (program counter), contents of the processor’s registers, process stack, data section containing global variables, and the text (the program) [SIL94]. The aforementioned description of a process is known as its data structure.

1. Process Data Structure

In general a process will have at a minimum a program or set of programs to execute, a set of data locations for local and global variables and all defined constants, a stack that is used to keep track of procedure calls and parameter passing between procedures, and a number of attributes that allow the OS to control the process [STA98]. The physical manifestation of a process or its data structure will depend on the platform.

First consider WINNT; the process in WINNT is known as the executive process (EPROCESS) [SOL98]. The EPROCESS block, Figure 2.1, contains attributes about the process as well as pointers to other data structures.

Two key data structures in the EPROCESS are the *kernel process* (KPROCESS) and the *process environment block* (PEB). The KPROCESS block is sometimes referred to as the process control block (PCB). The PCB contains information that the kernel needs to schedule threads. In WINNT the EPROCESS and KPROCESS both reside in system address space and the PEB resides in user address space. Access to system space, also known as kernel space, requires privilege. User space is accessible by non-privileged entities. The PEB contains the information needed to load the image loader, the heap manager and all Win32 DLLs that need to be modified from user mode.

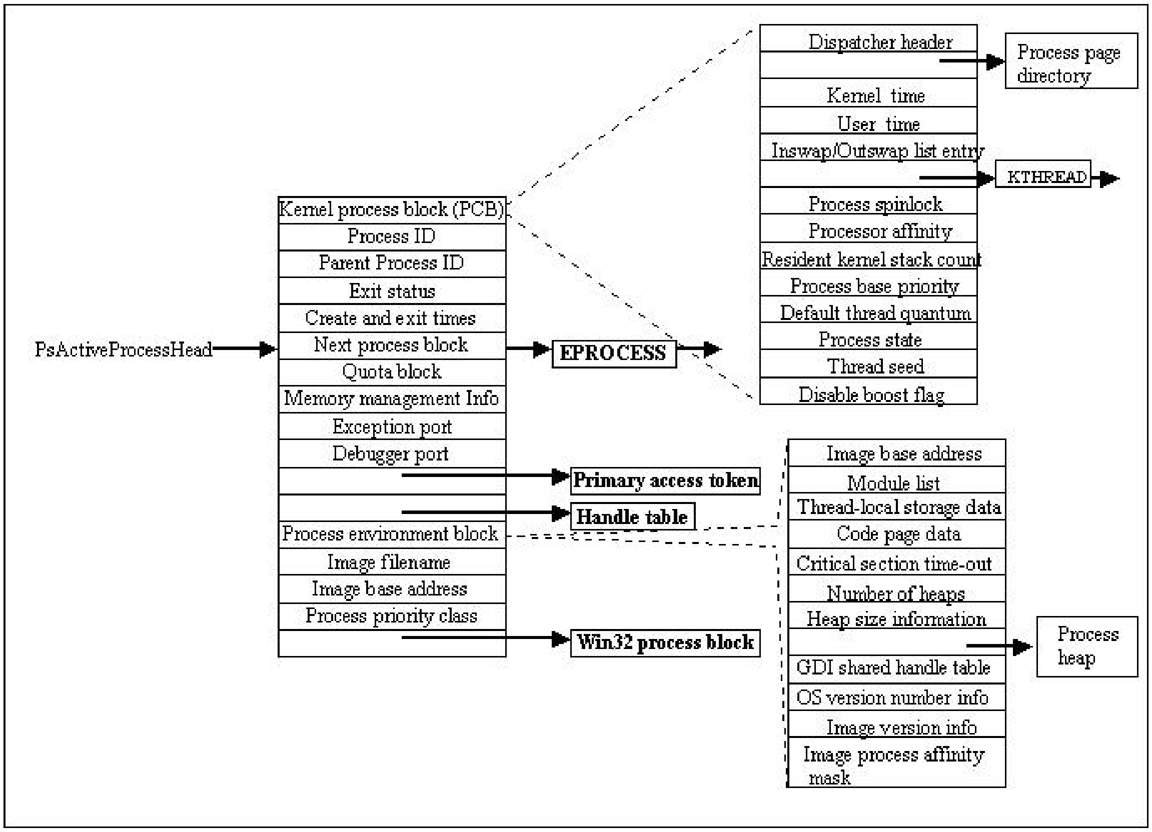

Figure 2.1 WINNT EPROCESS Block (After: [SOL98])

The image loader, heap manager and Win32 DLLs are user-level entities that the operating system uses to create and manage processes. During CreateProcess, a function call, the image loader determines the type of file associated with the request to run. Once the file type has been determined, the image loader loads the appropriate image for that file (notifies the appropriate subsystem of the new process); for example, appropriate images for Win32 processes are ***Posix.exe*** for POSIX file types and ***Ntvdm.exe*** for MS-DOS, .exe, .com, and .pif file types [SOL00]. If the file does not have an appropriate image, CreateProcess fails [SOL00]. The heap manager is a set of functions that controls the amount of memory allocated and deallocated [SOL00]. The Win32 DLLs are a set of callable subroutines in a binary file that are linked together and can be dynamically loaded by applications that use the subroutines [SOL00].

The UNIX process structure is now described. Figure 2.2 shows a Unix-based process's data structure. The kernel process table -- depicted in the figure as process table -- has an entry for each process. The "u area" contains the process's private data that can only be manipulated by the kernel and only when the specific process is

executing. The process table contains pointers to the per process region table. The entries in the *per process region table* point to entries in the *region table*. A region is defined as a contiguous area of a process's address space. The region table entries describes the attributes of the region, such as whether it contains text or data, whether it is shared or private, and where the "data" of the region is located in memory [BAC86].

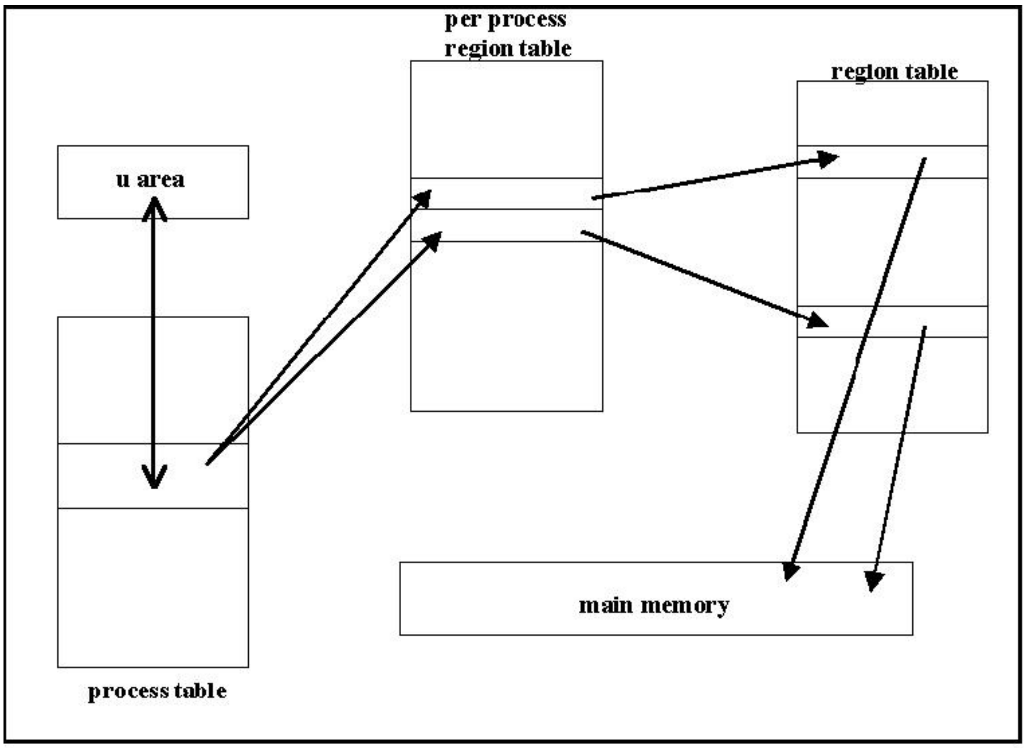

Figure 2.2 Unix Process Data Structure (From: [BAC86])

The data structure of a process, although it may be platform dependent in terms of its physical implementation, must meet the minimum requirements mentioned above: a program to execute, location of local and global variables, etc. This paper will not map the minimum requirements to each of the data structures; however, the reader should note that each data structure does implement and support the requirements. Each process managed by the OS has its own data structure independent of any other process managed by the OS. However, processes may share certain information and at times memory. This will be discussed further in Section F Subsection 1 of this chapter. Processes operate independently of one another and usually do not share memory, which is useful when the tasks to be performed are unrelated [SIL94].

2. Process Life Cycle

Another important aspect of a process is its life cycle. Processes make it possible for multi-tasking OSs running on a single processor to multi-task and make the computer appear to process multiple applications simultaneously. Processes primarily accomplish this through life cycle transitions and time allocation from the OS. There are several stages in the life cycle of a process. A complete process state transition diagram is depicted in Figure 2.3.

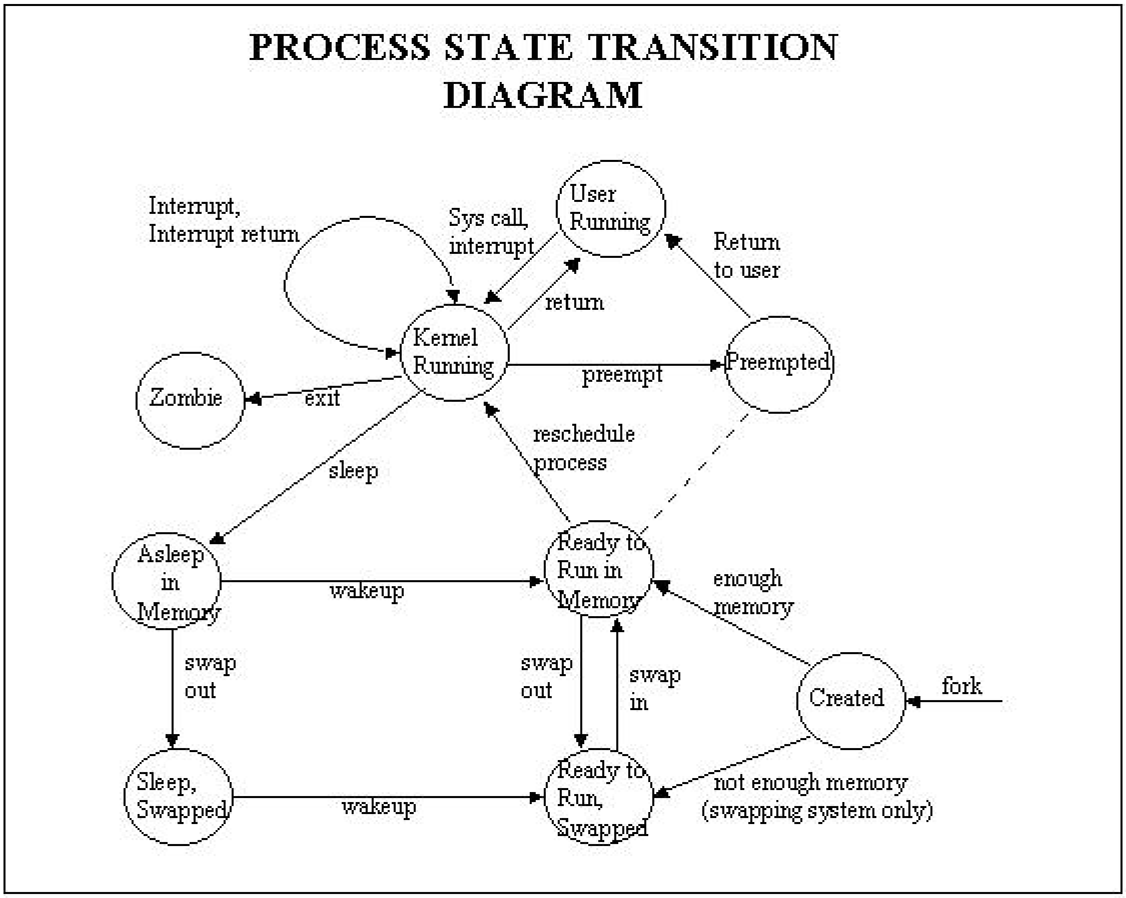

Figure 2.3 Process Life Cycle (From: [BAC86])

Figure 2.3 shows the transition state diagram for processes in a Unix-based OS. However, every process, regardless of the OS that supports it, can be represented in part or in full by the figure. Not every OS's supported processes will have the full array of transition states as shown in Figure 2.3. As a general rule, a process will always have one of the following states: new/created, running, waiting, ready, or terminated. Each transition state is defined as follows [BAC86, STA98]:

User running: the process is executing in user mode.

Kernel running: the process is executing in kernel mode.

Ready to run in memory: the process is not executing but is ready to run as soon as the kernel schedules it.

Asleep in memory: the process is sleeping and resides in main memory.

Ready to run, swapped: the process is ready to run, but the swapper (process 0) must swap the process into main memory before the kernel can schedule it to execute (although some [BAC86, STA98] refer to process 0 as a swapper, it may be more precise to think of it as a transitional mechanism that transitions a process from the **User running** state to **Ready to run in memory**, the process remains in main memory throughout the transition).

Sleep, swapped: the process is sleeping, and the swapper (swapper here refers to a mechanism that actually moves the process from main memory to secondary memory) has swapped the process to secondary storage to make room for other processes in main memory.

Preempted: the process is returning from kernel to user mode, but the kernel preempts it and does a context switch to schedule another process. The distinction between this state and the state "ready to run in memory" is based on the process. In this state a process running in kernel mode can be preempted only when it is about to return to user mode. Otherwise, the preempted state is the same as "ready to run in memory."

Fork: the process is newly created and is in the transition state; the process exists, but it is not ready to run, nor is it sleeping. This state is the start state for all processes except process 0.

Zombie: the process executed the exit system call and is in the zombie state. The process no longer exists, but it leaves a record containing an exit code and some timing statistics for its parent process to collect. The zombie state is the final state of the process.

There are several triggers that cause a process to transition from one state to another. One of the major triggers is central processing unit (CPU) time. How much CPU time the OS allocates to a process is important to the process. The time that is allocated to a process is known as a quantum. In a single-user, single task OS, processes execute sequentially, and once the process begins, it runs until it ends. In the single-user, single task system, the process has complete control of the CPU and resources until it has accomplished its task – the current process monopolizes the CPU until it completes its task. The user cannot move between various programs or open multiple windows.

In a single-user, multi-tasking operating system (common in most desktop operating systems today), processes must share the processor(s) and resources in accordance with a scheduling algorithm. Once a process is scheduled, it is given a quantum, in which it has exclusive use of the CPU and any unlocked resources. After the process's quantum expires, the OS, in accordance with its scheduling algorithm, gives the CPU and access to all unlocked resources to another process.

A multi-user, multi-tasking operating system, such as UNIX System V, sets up a computing environment for a user's process [STA98]. The multi-user OS must separate one user's resources from another's. Each user process has exclusive use of the CPU for a specified period of time, much like a quantum [STA98]. This method allows the OS to appear to execute application programs simultaneously, while the quantum and scheduling algorithm prevents processes from monopolizing the CPU, thereby preventing starvation or other detrimental situations for the OS.

3. Process Switching and Process Context

The method that allows the CPU in multi-tasking operating systems to move from one process to another in accordance with a scheduling algorithm is known as process swapping and sometimes it is referred to as process switching. A process must be in main memory to be executed. As mentioned earlier, to avoid starvation because one process controls or monopolizes the CPU, each process is given a quantum of time to have exclusive use of the CPU. Once the process's quantum expires, the process is removed from execution and another process is scheduled for execution.

Depending on the design of the operating system, there may be additional events, besides the expiration of the process's quantum that could cause processes to transition from the running state. Some of the more general events that are usually common to most OSs are preemption, which can be caused by process priority (one process having a higher priority than another), I/O operations, and traps (traps are for mode switching, not process switching, but a snap shot of the state of the machine (process or thread context) must be maintained during a trap [SOL98]). A process with a higher priority may preempt a lower priority process; note that many operating systems have a mechanism in place to ensure that low priority processes are not starved. I/O operations could cause a process to wait. In such cases, to optimize CPU usage, the process that is waiting on the I/O operation might transition from the running state.

A *trap* is a term that is used to explain a processor's mechanism for capturing an executing process or thread when an exception or an interrupt occurs which requires a snapshot of the process's context [SOL98]; the process's context is its registers, stack information, and other pertinent information, which needs to be preserved when the system mode is switching from user mode to kernel mode. The cause of a trap is usually associated with the execution of the current instruction [STA98]. Usually a trap is used to handle an error or an exceptional condition [STA98].

During process transitions the operating system must save the state and pertinent information of the currently executing process. The state and pertinent information of a process is the *process context*. In general (expanding the definition of the process context), it usually consists of the program counter, other process registers, and stack information from both user and kernel space (remember that a process has data, text, and a stack – the earlier definition of a process being an instance of a program corresponds with the "text" portion of the process). By freezing the process and capturing a snapshot of it as it exists in its halted or frozen state, the OS is able to move it from executing memory to storage and is later able to move it back into executing memory with its precise settings when it was frozen or halted. The OS prevents arbitrary process swapping and transitions thereby maintaining consistency.

In Unix-based operating systems, all processes except process 0 are spawned from other processes through the *fork* system call. Process 0 is created when the operating system boots up. Process 0 spawns process 1, which is known as *init*. The *init* process is the ancestor of every other process in the Unix system. A *parent* process is any process that spawns another process through the *fork* system call. The spawned process is known as the *child* process. Process 0 becomes the transitional mechanism. Both Unix-based OSs and NT-based OSs use a scheduling algorithm that schedules processes. However, WinCE, which is an NT-based OS, schedules threads. Threads will be discussed in the next section.

Each application (program) executing is assigned only one process by the operating system, the *parent process*. If other processes are required to assist the parent process in reserving resources to enable the program to execute, the additional processes are spawned by the parent process and not the operating system; however, each process is considered a separate execution sequence [SIL94]. For example, several users may be running copies of a mail program [SIL94]; in this example each user is assigned a *parent* process that works on their behalf and each mail program is a *child* process to a unique *parent* process, a one-to-one correspondence. In a similar example, a single user may invoke many copies of an editor program [SIL94]. In this example, each copy of the editor program is a *child* process to the same *parent* process, a one-to-many correspondence. A process can spawn one or more processes (children/child processes) or threads to assist in executing a program.

D. THREADS

For systems that support threads, the OS is designed to schedule threads instead of processes for a quantum. This is the case in Windows NT. A thread is an entity spawned from a process that executes a program [SOL98]. More simply stated, a thread is a precisely measurable controlled unit of work (a basic unit of CPU utilization) [STA98]. Threads enable resources (although threads do not reserve resources) to be shared and accessed concurrently within the same process, which is useful for related jobs. A thread can be spawned from a process or another thread. A thread can only spawn another thread -- a child thread. Threads allow a more granular definition of the dependencies

between processes and programs by completing the synergism that exists between the process and program. Many modern operating systems use processes and threads. This research will focus on Windows CE, which is a processed-based, single-user OS that schedules threads instead of processes.

Threads are very similar to processes. However, threads are used to enhance the capability of a process' multi-tasking ability. Threads allow resources to be shared and accessed concurrently within the same process by executing in the same address space (domain) [SIL94]. Threads are sometimes called lightweight processes [SIL94]. Threads generally consist of a program counter, a register set, and a stack space [SIL94]. A thread, like a process, is assigned to run in user space or system space depending on its privilege.

Threads operate similarly to processes in many regards in terms of their life cycle and context. Threads will generally have one of several states similar to processes: new/create, ready, blocked, running or terminated. Threads, unlike processes, are not independent of one another because they have to work together to share a pool of resources reserved for the process, whereas processes aren't required to work together in the same manner because each process has its own pool of reserved resources and the underlying OS provides the management.

Threads can be managed either at the application level or within the OS. Threads within a process execute sequentially and each thread has its own stack and program counter. Since Windows CE's scheduling algorithm schedules threads, threads will be covered in more detail in Chapter III. The Windows NT thread data structure will be discussed in detail in Section F Subsection 2 of this chapter. In addition to having a general understanding of threads and processes, enhancement of an OSs self-protection capability requires an understanding of security.

E. SECURITY

Security, which is usually thought of as privacy and protection, is often used in connection with information-storing systems. Security describes techniques that control access to use of or modification of a computer or the information contained in it. It is often useful to categorize security violations in three categories: unauthorized

information release, unauthorized information modification and unauthorized use [SAL75]. Chapter IV will take a closer look at security and how it can be applied to Windows CE, specifically in the case of authentication.

Security is based on three notions [MUL93]: authentication, access control and auditing. Authentication requires that for every request for an operation, the name of the requestor must be known reliably. The source of the request is called a principal. Access control requires that for every resource and every operation on that resource, it is possible to specify the names of the principles allowed to perform that operation on that resource. Every request for an operation is checked to ensure that the principal is allowed to perform that operation. Auditing permits every access to a resource to be logged if desired, and can be evidence used to authenticate every request. If a troubling situation occurs, there is a record of exactly what happened.

There are two ways to approach security. One approach involves constructing a formal specification for the desired security properties (and other properties) of a system then designing and implementing the OS. The other way is to find, catalog and repair security flaws in existing systems. This project will encompass both approaches. Security has its own terminology and it is broad. Scoping the security discussion and understanding the terminology make identifying and cataloging security flaws in the Windows CE authentication mechanism easier.

The scope of the security discussion for this paper is the authentication mechanism, modes of operation, and mechanisms that are common to processes. Windows CE operates in one of two modes: user mode or kernel mode. A mechanism is common to two processes if it uses some set of data items whose value one process can influence and the other can notice. Common mechanisms carry a built-in risk – they make it possible for the process of one user to exert unauthorized influence over the process or data of another. Malicious users can exploit flaws in common mechanisms to work their will.

According to Michael Schroeder [SCH75], the operating systems of shared general-purpose computers have a well-known tendency to be extraordinarily large and complex. This size and complexity interacts badly with the negative nature of security

requirements. It generates many possible ways to perform unauthorized actions, some of which will go unnoticed by system designers. This increases the probability of exploitable errors in the implementation of the provided protection mechanisms.

Examining the thread and process module of Windows NT will provide a basic understanding of the OS. Although the WINNT OS is substantially larger, with respect to lines of code than the Windows CE OS, WINNT affords some insight into the Windows-based platform. Windows CE, on the other hand, is a smaller system, and lacks many of the security features of Windows NT. Because WinCE lacks many of the security features found in WINNT, WinCE is more vulnerable. However, WinCE's lack of security features and increased modularity gives it the advantage of being less complex than WINNT, which naturally adheres to the design principle of economy of mechanism [SAL75].

Many systems try to reduce the probability of exploitable errors through good software engineering by utilizing well-established design principles to construct the internal mechanisms. These principles, when properly used, help to reduce design and implementation flaws, which might otherwise exist and provide paths by which the protection mechanisms in the system could be circumvented. Good engineering also utilizes established concepts such as the security kernel and reference monitor [AND72]. A security kernel is a minimal, protected core of software whose correct operation is sufficient to guarantee enforcement of the claimed constraints on access and is the structural basis for organizing a secure system [SCH75].

Although the goal is not to make Windows CE a high assurance secure system, it is prudent to maximize usage of the ideas and principles that govern secure systems. For example, while studying the design of a system, it is noticed that a mechanism in a more protected level of the OS depends on a function from a less protected or higher level of abstraction for the correctness of its operation. This is considered an upward dependency and a serious flaw in the system's design [GOL79]. It is also crucial to look at ways to implement security once a security policy has been established for the system.

Our goal in enhancing the Windows CE security is to balance security with performance. Since security by its very definition is an increase in system functionality,

i.e., the system's ability to a certain degree to protect itself and its information, performance will be adversely affected. Since Windows CE is a smaller OS on a small platform, it will have very limited space in which to implement security. Because WinCE has a more limited space to implement security than larger operating systems, the choice of which security features to implement is vital. If security is to work properly, care must be given when deciding what security features are required and what security mechanisms will be used to achieve the requirement. Security should be meticulously designed and carefully implemented.

For a particular mechanism or functionality, there may be a number of satisfactory implementations. Security implementations are described in terms of security concepts and mechanisms, such as: reference monitor, security kernel, capability lists (capabilities), access control lists (ACLs), discretionary access control (DAC), mandatory access control (MAC), domains, rings, and a host of additional concepts and mechanisms, too numerous to list. Not only do these concepts and mechanisms describe the type of security implemented, they also define the granularity of the security; for example, security can control authorized access at various levels such as: the entire system, directories, files, data within a files, compartments, etc. Another important aspect of security is knowing "who or what" has gained or is trying to gain access to the data, which is accomplished through the operating system [SAL75]. For authorized users of a system, this is normally accomplished through authentication. How Windows NT implements authentication is discussed in the next section.

F. WINDOWS NT AUTHENTICATION MECHANISM

This section will focus on the Windows NT authentication mechanism. However, since processes and threads are an integral part of the Windows NT OS and authentication mechanism, this section will discuss how processes and threads are created and how they communicate. *Fibers* will also be mentioned. These are a type of thread. In the discussion of the creation of processes and threads certain acronyms will be used. Such acronyms are standard in the OS literature and as much as possible this paper will try to minimize the use of unusual acronyms.

1. Windows NT-based Process and Thread Creation

Users as well as the OS manipulate processes through application programming interface (API) functions, but a process must first exist. A new process is created whenever an application calls the Win32 CreateProcess function. The creation of a Win32 process consists of several stages and involve three different parts of the OS: the Win32 client-side library KERNEL32.DLL, the Windows NT executive, and the Win32 subsystem process (CSRSS). Figure 2.4 summarizes the Win32 create process.

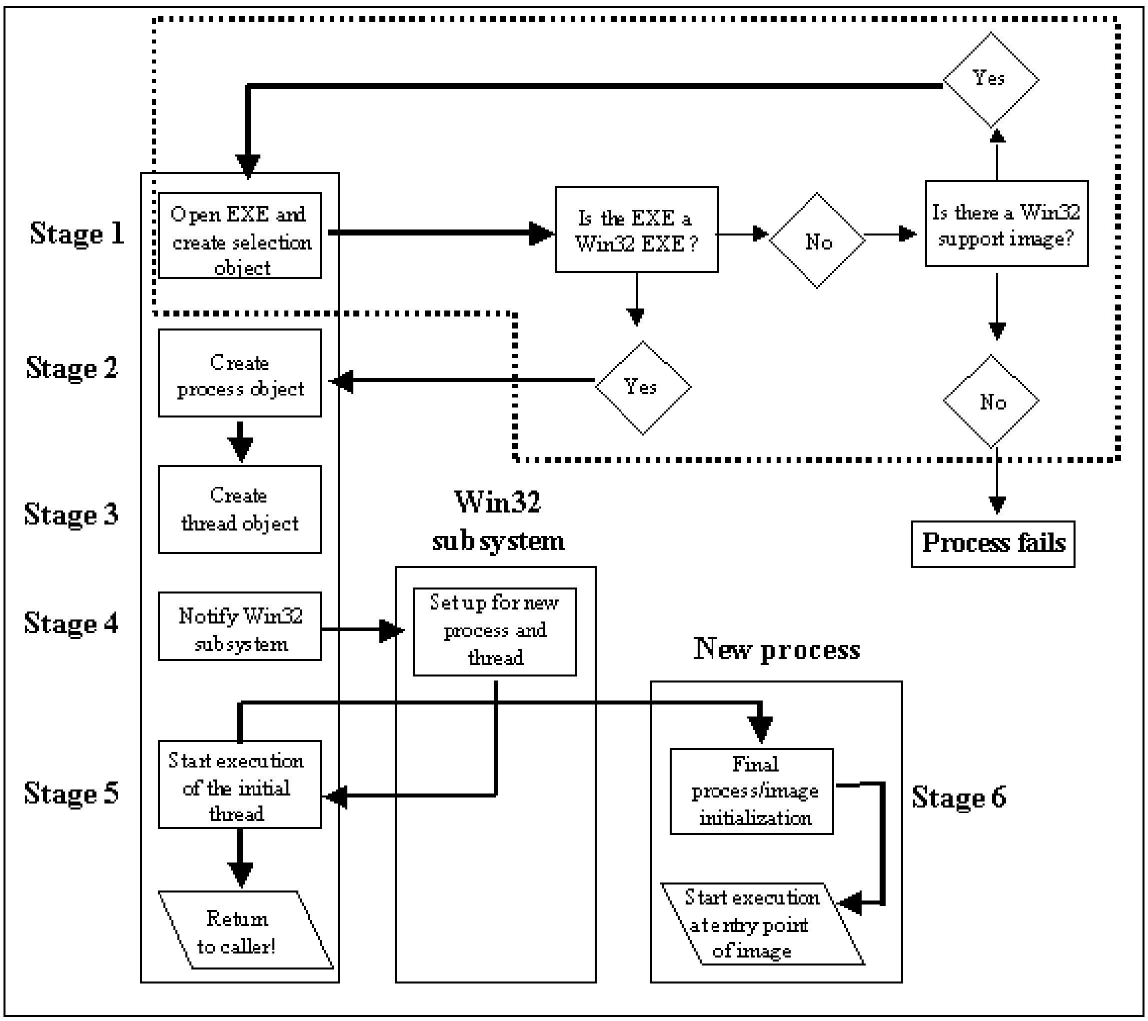

Figure 2.4 Process Creation (From: [SOL00])

A process's creation is complete as evidenced by the process environment having been determined, resources for its threads to use having been allocated, the process having a thread, and the Win32 subsystem having record of the new process. The new thread begins life running the kernel mode startup routine KiThreadStartup [SOL98].

KiThreadStartup performs thread-specific initialization, and then the actual thread routine specified by the caller of CreateThread is invoked [SOL98]. However, at this point, the thread has already been created, so the call to CreateThread is to notify the Win32 subsystem about the new thread so that the subsystem can perform some setup work for the new thread.

The initial thread's creation is initialized and completed during the Win32 CreateProcess call. In the CreateProcess, the call to create a thread filters down to the Windows NT executive, where the process manager allocates space for the thread object and calls the kernel to initialize the kernel thread block. At this point the Win32 CreateThread is called. The CreateThread creates a user-mode stack for the thread in the process's address space. CreateThread then initializes the thread's hardware context (this is CPU-architecture specific). The NtCreateThread is called to create the executive thread object in the suspended state. At this point, the CreateThread notifies the Win32 subsystem as mentioned earlier. The thread handle and ID are returned to the caller and unless the thread was created with the CREATE_SUSPENDED flag set, the thread is resumed so it can be scheduled for execution. Figure 2.5 summarizes the steps involved in thread creation.

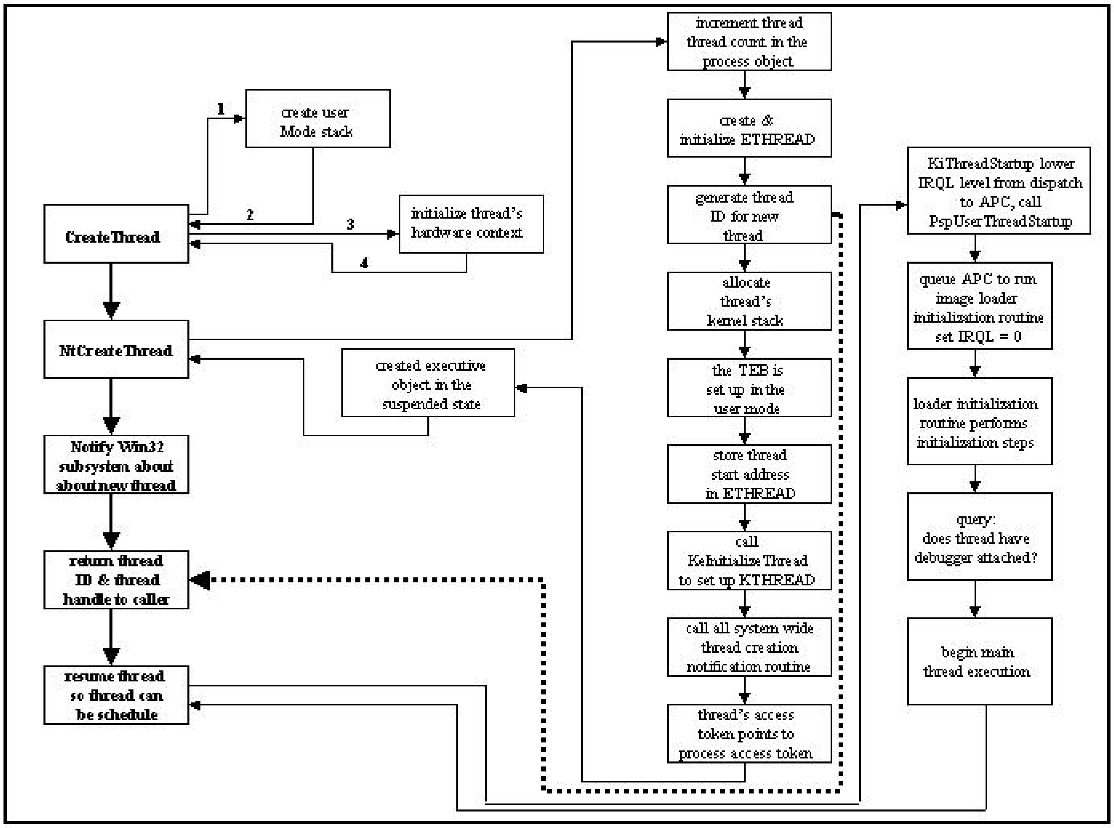

Figure 2.5 Create Thread

2. Windows NT Thread Data Structure

The executive thread (ETHREAD) block represents the Windows NT OS thread. The ETHREAD block contains two other thread data structures, the kernel thread (KTHREAD), which contains the information that the Windows NT kernel needs to access to perform thread scheduling and synchronization on behalf of running threads and the thread environment block (TEB), which stores context information for the image loader and various Win32 DLLs. The ETHREAD block and the structures it contains and points to exist in system space with the exception of the TEB, which exists in the process address space as shown in Figure 2.6.

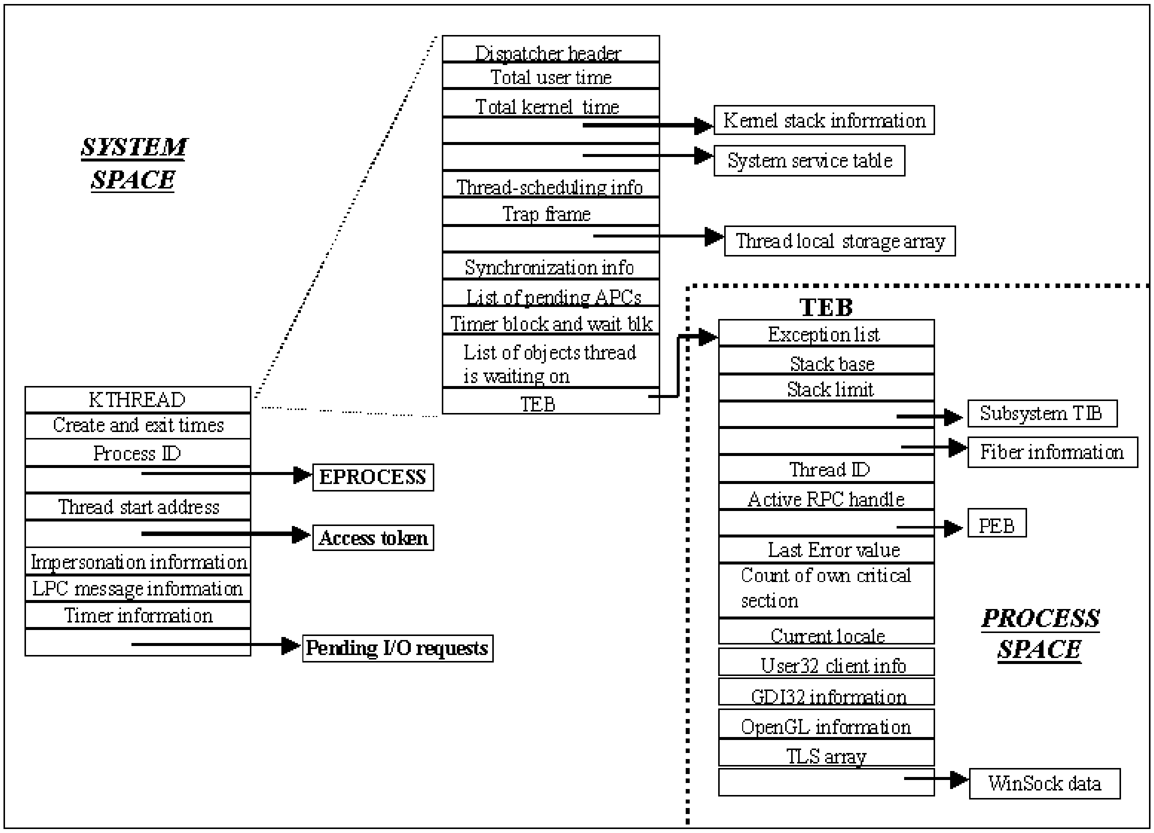

Figure 2.6 ETHREAD Block (After: [SOL98])

Although fibers are not a part of our research, it is important to understand fibers and their role in the Windows NT-based OS. Fibers are a subset of threads and are contained in the thread object. They are sometime referred to as “lightweight” threads. Fibers differ from threads in how they are scheduled. Threads are allocated CPU time by the OS. Fibers are not allocated CPU time by the OS; instead a programmer must manually schedule fibers to run. Fibers will not run unless the thread they are contained in is scheduled to run. Fiber scheduling is very simple: once the thread containing the fiber is scheduled to run, the fiber will run until it finishes or until it instructs the OS to run another fiber. Fiber functions were added to the Win32 API set primarily to support porting server applications from UNIX that were designed to schedule their own threads rather than relying on a priority system [SOL98].

The Windows NT priority system consists of 32 priority levels for threads ranging from 0 through 31. The range is divided into three groups. The groupings are real-time levels, which range from 16 through 31; variable levels, which range from 1 through 15;

and system level, which is 0. Threads are assigned a priority level from two different perspectives: the Win32 API and Windows NT kernel. The Windows NT kernel assigns the thread priority based on one of the 32 levels already mentioned. The Win32 API assignment involves the process's assignment. The Win32 API assigns each thread a priority based on a combination of the thread's process priority class and the thread's relative priority. The Win32 API process priority classes are "real-time," "high," "normal," and "idle." The Win32 thread's relative priorities are "time-critical," "highest," "above-normal," "normal," "below-normal," "lowest," and "idle." Table 2.1 is a mapping of the kernel thread priorities to the Win32 API priorities.

Every thread starts out with its priority equal to that of the process or thread that spawned it. Each thread has two priority values, its base priority, which is the priority that it starts with and its current priority, which is the priority at which the thread is currently running. A thread whose current priority is within the range 1 through 15 often has a lower base priority; however, a thread whose current priority is in the range 16 through 31 never has its priority adjusted by the OS, so the base priority is always equal to the current priority.

		Win32 process priority classes			
		Real time	High	Normal	Idle
Win32 thread priorities	Time critical	31	15	15	15
	Highest	26	15	10	6
	Above normal	25	14	9	5
	Normal	24	13	8	4
	Below normal	23	12	7	3
	Lowest	22	11	6	2
	Idle	16	1	1	1

Kernel priority levels

Table 2.1 Kernel Priority Levels (From: [SOL98])

Threads and processes are such an intricate part of the OS that they impact a system's ability to provide security for itself. Of specific interest is the WINNT

authentication mechanism. The authentication mechanism is initiated and driven by a process, the logon process (WinLogon). WINNT can be configured to F-C2/E3, which indicates more security than just an authentication mechanism, in accordance with the UK Information Technology Security Evaluation and Certification (ITSEC) board [SOL98]. (Note that ITSEC has been replaced in some countries by the international standard - The Common Criteria [CCI99].) Also, the U.S. Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) formally evaluated WINNT 3.51 at the C2 level [SOL98]. However, this research focuses on one of the first steps in creating a security-enhanced CE OS, which is providing a small, reliable authentication mechanism.

WINNT provides four basic security services [SOL98]: a secure logon facility, discretionary access control, security auditing, and memory protection. The secure logon facility requires users to identify and authenticate themselves by entering a unique logon identifier and password before they are allowed access to the system. There are several components that makes this secure logon possible: local security authority (LSA) server, security reference monitor (SRM), LSA policy database, security accounts manager (SAM) server, SAM database, default authentication package, and WinLogon. Each one is described below [SOL98].

Local security authority (LSA) server: A user-mode process running the image LSASS.EXE that is responsible for the local system security policy (such as which users are allowed to log on to the machine, password policies, the list of privileges granted to users and groups, and the system security auditing settings), user authentication, and sending security audit messages to the Event Log.

Security reference monitor (SRM): A component in the Windows NT executive (NTOSKRNL.EXE) that is responsible for performing security access checks on objects, manipulating privileges (user rights), and generating any resulting security audit messages.

LSA policy database: A database that contains the system security policy settings. This database is stored in the registry under HKEY_LOCAL_MACHINE\Security. It includes such information as what domains are

trusted to authenticate logon attempts, who has permission to access the system and how (interactive, network, and service logons), who is assigned which privileges, and what kind of security auditing is to be performed.

Security accounts manager (SAM) server: A set of subroutines responsible for managing the database that contains the usernames and groups defined on the local machine or for a domain (if the system is a domain controller). The SAM runs in the context of the LSASS process.

SAM database: A database that contains the defined users and groups, along with their passwords and other attributes. This database is stored in the registry under HKEY_LOCAL_MACHINE\SAM.

Default authentication package: A dynamic-link library (DLL) named MSV1_0.DLL that runs in the context of the LSASS process that implements Windows NT authentication. This DLL is responsible for checking whether a given username and password match what is specified in the SAM database, and if they do, returning the information about that user.

Logon process: A user-mode process running WINLOGON.EXE that is responsible for capturing the username and password, sending them to the LSA for verification, and creating the initial process in the user's session.

These components work together to provide logon authentication. The security reference monitor (SRM) is the only component located in kernel space. The LSA, which is in user space, is responsible for communicating with the SRM. They communicate using local procedure calls (LPC). The SRM creates a port (*SeRmCommandPort*), during system initialization. The LSA connects to that port. The LSA creates an LPC port (*SeLsaCommandPort*) when the LSA process starts. The SRM connects to *SeLsaCommandPort*. During system initialization, once the respective processes have connected to the other's created port, the result is a private communication port and neither process listens on their respective connect port any longer. Since the processes no longer listen to their respective port and the ports are unnamed, a subsequent user process cannot successfully connect to either port (if the

entry points still exist after the initialization phase, they may be exploitable; however, this is difficult to determine without inspection of the SRM code).

The WinLogon process is the only process that can intercept logon requests from the keyboard. During system initialization, WinLogon is given control of the workstation once the system is ready for user interaction [SOL00]. WinLogon creates three desktops, an application desktop, a WinLogon desktop, and a screen saver desktop [SOL00]. Only WinLogon can access the WinLogon desktop making the WinLogon process a trusted process.

If a user wants to access the system (legitimately), the user must go through the WinLogon desktop. The WinLogon desktop will authenticate the user. It communicates with the LSA, using an LPC, to authenticate the user. Once the user has been authenticated, the user is given access to the application desktop and the screen saver desktop. A concern with this logon model is its lack of simplicity; its GUI code required to logon is enormous in contrast to the XTS 300 logon interface, which is not ideal for users but simple. The UI for the XTS300 is not user friendly nor is it an intuitive "point and click" interface. It requires the user to understand the various sessions levels for logon and what privileges the user will have at the various levels. It is simple because the code to present this interface can be minimized.

G. CONCLUSION

Processes are essential to computer usage, from system initialization, user authentication, to system shutdown. Understanding the process's requirements for system resources for program execution and how operating systems manage those resources gives us a better understanding of the importance of local (single desktop) computer security within a large network of computers. As mentioned at the beginning of the chapter, computers are playing an increasingly larger role in our everyday lives. In networks, a compromised process (compromised program that a process is executing) on a single computer could affect other computers in that network as well. Understanding processes and their roles in OSs is important to the design of the OS as well as security. Now that a better appreciation and understanding of operating systems (specifically

Windows NT), threads, processes and security has been gained, Chapter III can focus on Windows CE threads and processes.

III. WINDOWS CE PROCESSES AND THREADS

A. INTRODUCTION

Processes are a vital part of program execution as discussed in Chapter II. Chapter II also described an operating system as a program. Therefore, process management is a vital role of the operating system. This chapter will focus on the processes and threads in the WinCE operating system.

Windows CE is a multithreaded WIN32 system, with the same process and thread model and file formats as Windows NT (WINNT) [MUR98]. In both cases, the kernel supports multitasking; it has preemptive, priority-based scheduling [MUR98]. Windows CE has characteristics in common with WINNT. Some of the complexities of the WINNT process model have been removed from Windows CE, but the "animated spirit" (model) has been left in tact.

This chapter describes Windows CE's "animated spirit," processes, in more detail. Processes and threads in the WINNT-based OS have been defined. Their data structures have been discussed, and their life cycle has been examined. However, because Windows CE is a smaller OS that operates in a smaller memory space, there are some differences between its processes and threads as compared to those of WINNT.

B. WINDOWS CE

What is Windows CE and why is it important? A white paper [MWS2a] from the Microsoft Corporation provides answers to the question. It states:

> Microsoft Windows CE is an open, scalable, 32-bit operating system that is designed to meet the needs of a broad range of intelligent devices, from enterprise tools such as industrial controllers, communications hubs, and point-of-sale terminals to consumer products such as cameras, telephones, and home entertainment devices. A typical Windows CE-based embedded system targets a specific use, runs disconnected from other computers, and

> requires an operating system that has a small footprint and a built-in deterministic response to interrupts.
>
> Windows CE consists of a set of discrete modules and sub modules, or components, each of which provides full or partial support for major features of the operating system. By selecting a minimum set of modules and components, a device manufacturer can design an operating system tailored to requirements of a particular device. By controlling the size ("footprint") of the operating system, original equipment manufacturers (OEMs) can design for speed and efficiency, while still providing the performance of 32-bit, preemptive multitasking, multithreaded system and the richest possible set of APIs for developing applications. [MWS2a]

Microsoft designed CE to compete in the embedded systems market [MUR98]. Microsoft had a specific class of embedded devices in mind for CE. Instead of being stand-alone devices, these devices are intended to be mobile companion devices that supplement the desktop and synchronize information with the desktop personal computer (PC) [MUR98]. Approaching the design of the CE operating system with the idea that it supports a companion device may explain why Windows CE's processes and threads maintain the look and feel of the WINNT's processes and threads.

Unlike Windows NT, Windows CE can only support up to 32 simultaneous processes [MIC98]. When WinCE initializes, it creates a single 4GB virtual address space; it divides that 4GB address space in half creating (2) 2GB spaces, and it further divides 1 of the 2GB address spaces into 33 equal slots ranging from 0 – 32. When a process is initialized, WinCE selects an open slot in 1 of the allocated 33 slots [MIC98]. Slot 0 is always reserved for the currently running process [MIC98].

Windows CE does not assign processor time to processes, as does WINNT, instead Windows CE assigns processor time to threads. In WinCE, threads are the unit of execution instead of processes, and are assigned execution quanta by the scheduler [MIC99]. This means that a process is terminated if its primary thread is terminated. Although it is the threads that are assigned quanta, it is the processes that reserve the resources necessary to enable users to open and work in several applications at the same time [MIC99].

C. WINDOWS CE PROCESS

Only 32 processes can execute simultaneously in WinCE, and, not all 32 slots are available to the user [MWS2b]. A minimum of four, and more typically seven or eight, processes are created when the system starts up. These are system processes not directly accessible to users [MWS2c]. WinCE will usually utilize process slots for the following processes at system start up (system processes): kernel (nk.exe), device drivers (device.exe), installable file systems (filesys.exe), and graphical user interface, GUI, (gwes.exe) [MWS2b]. WinCE process's address space, 32MB, which accounts for the limited number of processes that WinCE can simultaneously execute, is 64 times smaller than the address space for processes in the desktop version of Windows, usually 2048MB [MWS2b].

The WinCE process data structure has the following fields: a virtual address space, executable code, data, object handles, environment variables, base priority, and minimum and maximum working set sizes [MWS2d]. Processes are created by applications, or by the shell when a program is invoked by the user (such as double clicking a program icon) [GAR99]. Creating a process is done by calling *CreateProcess()* [GAR99]. *CreateProcess()* creates a new process and its primary thread; it is a function used to run a new program [MWS2e]. Figure 3.1 demonstrates how process creation is invoked.

```
void foo()
{
    PROCESS_INFORMATION    ProcInfo;    // Process information structure
    BOOL                   rc;

    rc = CreateProcess (
            Text ( "Hello.exe" ),           // Application name
            NULL,                           // Command line arguments
            NULL,                           // No process attributes on CE
            NULL,                           // No thread attributes on CE
            FALSE,                          // No handle inheritance on CE
            0,                              // No special flags
            NULL,                           // No environment on CE
            NULL,                           // No current directory on CE
            NULL,                           // No startup info on CE
            &ProcInfo);                     // Get info about that process

    // rc is TRUE if the process has been created, FALSE otherwise

    ………………..
}
```

Figure 3.1 CreateProcess Function (From: [GAR99])

1. Creating a Process

Once the CreateProcess( ) function is executed, the new process executes the specified executable file [MWS2e]. The following parameters (which are not part of the *Process_Information* structure) are a part of the CreateProcess( ) function call [MWS2e]:

lpszImageName

The lpszImageName parameter is a pointer to a null-terminated string that specifies the module to execute. The string can specify the full path and filename of the module to execute or it can specify a partial path and filename if the program is in the current working directory. The lpszImageName parameter must be non-NULL and must include the module name.

lpszCmdLine

The lpszCmdLine parameter is a pointer to a null-terminated string that specifies the command line to execute. The lpszCmdLine parameter can be NULL. In that case, the function uses the string pointed to by lpszImageName as the command line.

If both lpszImageName and lpszCmdLine are non-NULL, * lpszImageName specifies the module to execute, and * lpszCmdLine specifies the command line, (note that * indicates pointer). C language runtime processes can use the argc and argv arguments. If the filename does not contain an extension, .EXE is assumed. If the filename ends in a period (.) with no extension, or the filename contains a path, .EXE is not appended. Windows CE versions 2.10 and later search the directories indicated by the lpszImageName parameter, if the path of the file being looked for is not explicitly specified, in the following order:

The windows (\windows) directory

The root (\) directory of the device

An OEM-dependent directory

The OEM-defined shell (\ceshell) directory — this is available to Platform Builder users only

lpsaProcess

Not supported; set to NULL.

This is a security attribute that determines how the new process will be shared. It controls what another process may do if the other process opens a duplicate handle.

lpsaThread

Not supported; set to NULL.

This is a security attribute that determines how the new thread will be shared. It controls what another thread may do if the other thread has a duplicate handle.

fInheritHandles

Not supported; set to NULL.

This is a Boolean value that determines whether or not the new process will inherit other handles.

fdwCreate

The creation flags in the *fdwCreate* parameter govern the type and priority of the new process. The following creation flags can be specified in any combination, except as noted in Table 3.1:

Value	**Description**
CREATE_DEFAULT_ERROR_MODE	Not supported.
CREATE_NEW_CONSOLE	The new process has a new console, instead of inheriting the parent's console. This flag cannot be used with the DETACHED_PROCESS flag. The console is a component of the command processor shell. On the IPaqs, the component used is explorer, a Handheld PC style shell.
CREATE_NEW_PROCESS _GROUP	Not supported.
CREATE_SEPARATE_WOW_VDM	Not supported.
CREATE_SHARED_WOW_VDM	Not supported.
CREATE_SUSPENDED	The primary thread of the new process is created in a suspended state, and does not run until the **ResumeThread** function is called.
CREATE_UNICODE_ENVIRONMENT	Not supported.
DEBUG_PROCESS	If this flag is set, the calling process is treated as a

	debugger, and the new process is a process being debugged. Child processes of the new process are also debugged. The system notifies the debugger of all debug events that occur in the process being debugged.
DEBUG_PROCESS	If you create a process with this flag set, only the calling thread (the thread that called **CreateProcess**) can call the **WaitForDebugEvent** function.
DEBUG_ONLY_THIS_ PROCESS	If this flag is set, the calling process is treated as a debugger, and the new process is a process being debugged. No child processes of the new process are debugged. The system notifies the debugger of all debug events that occur in the process being debugged.
DETACHED_ PROCESS	Not supported.

Table 3.1 Creation Flags for CE CreateProcess (From: [MWS2e])

Windows CE does not support the concept of priority classes for processes. The priority of a thread is the only parameter that determines a thread's scheduling priority.

lpvEnvironment

Not supported; set to NULL.

Environment variable, generally used to help customize programs and usually kept where they (variables) are always available.

lpszCurDir

Not supported; set to NULL.

Current directory setting

lpsiStartInfo

Not supported; set to NULL.

This is a structure that the parent process must fill out before calling *CreateProcess*.

lppiProcInfo

The lppiProcInfo parameter is a pointer to a PROCESS_INFORMATION structure that receives identification information about the new process.

This section, Section C, began by introducing the WinCE process, specifically the process's requirements – virtual address space, executable code, data, object handles, environment variables, base priority, and minimum and maximum working set size. This section also discussed the CreateProcess function, which detailed the necessary parameters for creating a process. However, this section did not define the WinCE data structure, but, with the given information and knowing that the WinCE process and thread model is the same as for WINNT, it is conceivable that the data structures are similar. By mapping the WinCE process requirements and WinCE CreateProcess function into the WINNT EPROCESS block (Figure 2.1), an approximation of the WinCE data structure is obtained, which is equivalent to Figure 2.1. Table 3.2 and 3.3 map the CE process requirements and CreateProcess function parameters to the WINNT EPROCESS block.

WinCE process requirements	WINNT EPROCESS block
Virtual address space	Memory management info
Executable code	Process environment block
Data	Process environment block
Object handles	Handle table
Environmental variables	????
Base priority	Kernel process block
Min & max working set size	Memory

Table 3.2 Mapping of Process Requirements to EPROCESS

CreateProcess function parameters	WINNT EPROCESS block
LpszImageName	ImageFileName
LpszCmdLine	????
fdwCreate	Process priority class not supported in CE
lppiProcInfo	????

Table 3.3 Mapping of *CreateProcess* Function Parameters to EPROCESS

Windows CE operates in one of two modes (as mentioned earlier), user mode or kernel mode, and for the sake of performance, usually user mode is not invoked (so everything runs in kernel mode); so the areas that the CE data structure occupies may be different than that of the WINNT process data structure. For instance, in the WINNT data structure, the KPROCESS and EPROCESS are both in kernel space. It makes sense to separate the various parts of the process data structure for the WINNT process because some of its information and manipulations requires protection or privileged access. Since

OEMs develop their software and applications to run in privileged mode, and since there is no separation of the address space within CE, there is no benefit for the various parts of the process to occupy different space based on their need for privileged access. Indeed, the limited address space may not support such a design. Remember each process is initially allocated 32MB of memory. All other memory comes from a common pool that is shared by all processes, 2GB.

Although the process's data structure in CE is similar if not the same as that of WINNT, it does not support all the same functionality as the WINNT process. This in itself is no great revelation given the smaller OS, but it does raise the question of extensibility. Does this process data structure support the ability to be extended for future needs? The answer to the question is beyond the scope of this thesis, but it is one that needs to be considered. The process data structure of CE appears to have been designed with the ability to handle applications that can run on the larger NT-based operating systems, but each application is subject to the memory constraints of CE.

2. Terminating a Process

Process termination is a subject usually overlooked or deemed insignificant in larger operating systems because the effect of one process not terminating properly, especially if it is not a critical process, can be relatively insignificant. The process may hang or terminate in some non-desirable fashion, i.e. the user may see an exception message or a page fault message. With the exception of a message, the termination of a process may not affect system performance (in larger operating systems) to any noticeable degree and usually will not affect the number of processes that a user can execute, at least from the average user's perspective. However, terminating a process in CE is important because, given only 32 slots available for processes, its direct effect on the number of processes that can be executed will be more noticeable to the user.

Unlike the larger operating systems, CE cannot afford to have a process hang. In CE, the impact of a hanging process on performance will be immediately apparent to the user. Not only does a hanging process's affect the number of processes that can run, and possibly the memory, but it also puts a drain on precious battery power. Processes whether in larger NT-based operating systems or in CE have similar terminating

functions [MWS2f]. In WINNT a process can exist if its primary thread is terminated; however, in WinCE if a process's primary thread is terminated, the process is automatically terminated.

D. WINDOWS CE THREADS

If the process of the Windows CE operating systems is equated with the "spirit" of the OS, the thread is the heart and soul of the OS. It is the thread that gives life to the process. Each process is started with a single thread, the primary thread, and the process can create additional threads from any of its threads [MWS2d]. The number of threads that can be created in a process is limited only by the amount of RAM available [MWS2h]. Each thread belonging to a particular process runs in the context of that process. Also, all the threads of a process share its virtual address space [MWS2d]. A thread can execute any part of the process code to which it belongs, including parts currently being executed by another thread; and it is the thread that is allocated processor time (quantum) [MWS2g].

Windows CE version 3.0 and higher offers 256 priority levels for threads [MWS2c]. The highest priority level is 0 and the lowest is 255. Applications use levels 255 through 248, which maps to the 8 priority levels available in earlier versions of Windows CE. Real-time applications, drivers, and system processes use levels 247 through 0. Windows CE does not support process priority classes, which means theoretically that a process could hold a thread with the very highest level and a thread with the very lowest level [MWS2b/c]. A system scheduler determines which threads should run and when they should run. A process starts when the scheduler gives execution control to one of its threads [MWS2c]. Threads with higher priorities run before threads with lower priorities, and threads that have the same priority run in a round-robin fashion. The threads' quantum has a default value of 25 milliseconds, which the OEM can set to a different value.

Priority inversion, which occurs when a high priority thread is blocked from running because a lower priority thread owns a kernel object, is handled differently in WinCE than in other Windows operating systems [MWS2c]. Other Windows operating systems will raise all the low priority threads in the chain all at once, whereas CE will

raise each thread in the chain one at a time as necessary until the lower priority thread relinquishes the resource that the high priority thread needs. WinCE guarantees the handling of priority inversion only to a depth of one level; for example, if thread 1 (a high priority thread) blocks waiting on a critical section held by thread 2 (a lower priority thread), thread 2 is raised to the same priority as thread 1. However, if thread 3 is also in that chain, it will remain at its current priority level. In other Windows operating systems, thread 3's priority would also be raised.

Threads in WinCE have a much greater role than those in other Windows operating systems. The WinCE OS functionality is driven by threads and not processes. This does not necessarily mean that the data structure of the thread in WinCE needs to be different than that of WINNT. But, it does change how the operating system functions. Thread creation is the same for both NT-based operating systems and WinCE. Both operating systems use the *CreateThread* function.

1. Creating a Thread

The *CreateThread* function requires six parameters. Two of the parameters are set to NULL, *lpThreadAttributes* and *dwStackSize*. The parameter *lpThreadAttribute* is not supported by WinCe. The parameter *dwStackSize* is automatically set. The number of threads that can be created depends on the default stack size [MSW2i]. The default stack size for committed and reserve memory is specified in the executable file header [MWS2t]. A returned handle to a new thread indicates success and a NULL value indicates failure. The *CreateThread* function is described below [MSW2i]:

HANDLE CreateThread(

LPSECURITY_ATTRIBUTES	*lpThreadAttributes,*
DWORD	*dwStackSize,*
LPTHREAD_START_ROUTINE	*lpStartAddress,*
LPVOID	*lpParameter,*
DWORD	*dwCreationFlags,*
LPDWORD	*lpThreadId* **);**

lpThreadAttributes

Ignored. Must be NULL.

This is a pointer to a structure that specifies a security descriptor for the new thread and determines whether child processes can inherit the returned handle.

dwStackSize

Ignored. The default stack size for a thread is determined by the linker setting /STACK.

Specifies the initial commit size of the stack, in bytes.

lpStartAddress

Long pointer to the application-defined function of type LPTHREAD_START_ROUTINE to be executed by the thread and represents the starting address of the thread, points to the start of the thread routine.

lpParameter

Long pointer to a single 32-bit parameter value passed to the thread.

Specifies an application-defined value that is passed to the thread routine.

dwCreationFlags

Specifies flags that control the creation of the thread are shown in Table 3.4.

Value	Description
CREATE_SUSPENDED	The thread is created in a suspended state, and will not run until the *ResumeThread* function is called.
0	The thread runs immediately after creation.

Table 3.4 Control Flags for Creating a Thread (From: [MSW2i])

lpThreadId

This parameter is a long pointer to a 32-bit variable that receives the thread identifier. If this parameter is NULL, the thread identifier is not returned.

The *CreateThread* function describes the creation of a thread, but it should not be confused with the data structure of a thread. Just as with the processes in WinCE, there is no evidence to indicate that the thread in WinCE has a different data structure than the threads in WINNT. There are parameters, as shown in the function above, in the *CreateThread* function in WinCE that are not supported. But, the non-supportability of those parameters does not impact the data structure. The non-supportability impacts the user's ability to control specific fields in the data structure. Reference Figure 2.6 to recall the thread's data structure in WINNT.

2. Terminating a Thread

Terminating a thread is as important as creating a thread. As previously discussed, the proper functioning of the WinCE OS is linked to the proper functioning of its threads, not its processes. Therefore, it is vital that the OS not only knows which thread to terminate, but also which thread termination function to use. Terminating the wrong thread, for example the primary thread, prematurely could have undesirable consequences, such as terminating the process and any other threads associated with that process. This would not only be an annoyance to the user, but it would render the OS unsuitable because the OS might randomly shut down programs thereby creating instability. Some of these instabilities will be discussed below.

The operating system must know which thread termination function to use when terminating a thread. There are several functions that can be used when terminating a thread in WinCE [MWS2j]. If possible, the *TerminateThread( )* function should be avoided; but if used, it must be used very carefully because it has dangerous side effects [MWS2j]. The side effects are as follows [MWS2j]:

> If the thread targeted for termination owns a critical section, the critical section will not be released.

> If the thread targeted for termination is executing certain kernel32 calls when it is terminated, the kernel 32 state for the thread's process could be inconsistent.
>
> If the targeted thread is manipulating the global state of a shared DLL, the state of the DLL could be destroyed, affecting other users of the DLL. [MWS2j]

The side effects from the TerminateThread( ) function could possibly lead to an unstable operating system. Therefore, programmers and designers must take special care when writing applications that run on WinCE, not only with normal user applications, but also with system applications.

One such system application of concern is the authentication protocol. An authentication protocol represents the first step in developing a security enhanced WinCE OS. Talisker, which is the current version of WinCE in development, uses Kerberos as an authentication protocol.

E. CONCLUSION

Whether or not Kerberos is the best protocol to use in WinCE is a question this research will attempt to answer. Chapter IV will discuss various authentication protocols. The authentication protocol must be able to work with Microsoft's ".Net" (pronounced dot net) project, for obvious reasons. Most importantly, the protocol must be small, correct and efficient. As each protocol is discussed in Chapter IV, several questions must be asked:

Will this protocol work well with threads?

How many process slots does this protocol need to execute, including supporting processes?

Once these questions have been answered, enhancing the security in WinCE will still be a challenge. However, knowing how the operating system manages processes and threads, in particular how many process slots are available and that the threads are allocated process time instead of the processes is imperative to understand the system sufficiently to provide any successful security enhancement.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. AUTHENTICATION

A. AUTHENTICATION BACKGROUND

Authentication provides evidence to one party (Bob) that another party (Alice) is who she says she is. People are constantly authenticating in their day-to-day interaction with their surroundings. People use a variety of means to authenticate one another as well as other objects. People can be viewed as complex mechanisms that use a complex set of authentication protocols (senses) and rules to authenticate animate and inanimate objects. This complex set of protocols and rules is not yet available to computers. But, as technology continues to advance, computers may one day be able to use a similar set of complex protocols to authenticate various types of objects as well. Until then, developers must use the most appropriate and most robust authentication protocol available.

Examples of how different authentication protocols work best for different situations and environments can be observed in people's daily authentication processes. In this example, Alice uses the most appropriate "sense" (authentication protocol) to verify whether or not Bob is who he claims to be. If Alice is walking down the street and thinks she hears Bob's voice calling her from a short distance saying, "Hey Alice it's me Bob," she might turn in the direction where the voice originated. Upon turning, Alice immediately shifts to another protocol to verify whether or not it truly is Bob she hears calling her. At this point, although Alice's sense of hearing initially picked up on an object that sounded like Bob, her sense of hearing was not the most appropriate protocol in the given environment to authenticate Bob; Alice instead used her sense of sight to confirm whether or not the object was Bob.

In a different environment, the protocol might change as well. For example, if Bob calls Alice on the telephone, upon picking up the receiver and acknowledging a successful transmission connection, Alice immediately listens to the tone and inflection

of the voice emanating from the receiver. Once Bob identifies himself, Alice uses her sense of hearing to decide whether or not she believes the person on the other end of the connection really is Bob. Alice's sense of hearing is no more reliable in this situation than in the previous situation, but it is more appropriate. There may be times when Alice will use multiple protocols to authenticate an object; and there may be times when Alice is not equipped with the proper set of protocols to authenticate a given object. However, in most situations, Alice can feel reasonably sure she can authenticate most objects in most environments; computers are not so lucky.

Not every computer's operating system will support every authentication protocol. When multiple authentication protocols are supported by an OS (as is the case with Windows 2000 Server [BOS00]), the protocol to be used will be determined by both the client and server (based on protocols common to both) [SOL00]. Furthermore, it is assumed that having multiple authentication protocols not only adds complexity to the system, but it also requires more memory from the system. Complexity and increased memory requirements are contrary to maintaining the smallest possible footprint for WinCE. This chapter will discuss several authentication protocols supported in Talisker, and the Department of Defense's Public Key Infrastructure (PKI).

Talisker, the next generation WinCE operating system, presumably should provide more functionality than its predecessor. One function that Talisker provides that its predecessor does not is support for Kerberos. Kerberos is an authentication protocol used for mutual authentication between a client and server. [OWS3n] Kerberos is also supported by Windows 2000 [MWS2v].

B. KERBEROS

Conceptually, Kerberos is an authentication protocol based on tickets and session keys. In a network scheme, in order for Alice to prove her identity to Bob, she first has to obtain a ticket from a centralized authority to present to Bob. The centralized authority is known as the **K**ey **D**istribution **C**enter (KDC). Domain controllers in Windows 2000 implement the KDC. [MWS2v] Windows 2000 is mentioned and used to keep the example simple.

Each domain contains a Kerberos authority, which a user (client) of that domain must contact to receive a ticket from the KDC. If Alice and Bob are in the same domain, they share the same Kerberos authority. In the case where Alice shares the same domain as Bob, Alice can go directly to her local KDC and get a ticket for Bob. To get a ticket for Bob, Alice must prove her identity to the KDC by proving knowledge of her password or private key (in the case of smart card logon). [MWS2v]

The ticket that Alice gets contains a lot of information, which includes Alice's name, and a randomly generated encryption key (session key). The KDC locks this information in the ticket by encrypting it with a master key generated from Bob's password that only Bob and the KDC share. Once Alice presents the ticket to Bob, Bob can verify that the contents really came from the KDC by successfully decrypting the ticket. In this scheme the idea is that Bob will decrypt the ticket, look at the client's name, and realize that whoever sent the ticket to him must really be Alice because the KDC validated her identity before giving her the ticket. [MWS2v]

The reader should note in the previous example, Bob really has no idea who sent him the ticket. Although understanding the ticket process in Kerberos is fairly straight forward, there is another step included in the process to ensure that a bad guy is not simply replaying a recorded version of the ticket and masquerading as Alice. This additional step requires Alice to send the ticket and prove that she is the owner of the ticket. This is accomplished through the use of the session key. [MWS2v]

Alice must prove to Bob that she knows the session key that the KDC gave to both Bob and her. Alice proves to Bob she knows the session key by sending him a little package called an authenticator along with the ticket. The authenticator is Alice's name plus the time on her clock, encrypted with the session key. If Bob can decrypt this authenticator, the name inside matches the client's name inside the ticket, and the timestamp is recent and not in his replay cache, Bob accepts this as proof of Alice's identity. [MWS2v]

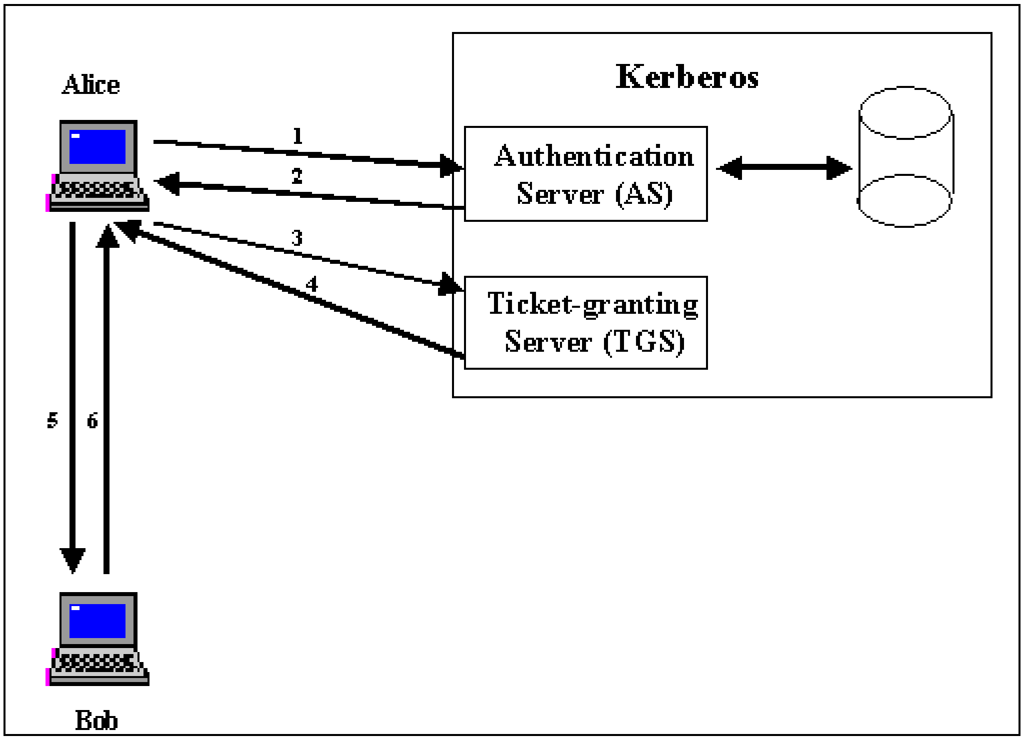

Figure 4.1 Kerberos Protocol Process (After: [STA99])

Figure 4.1 depicts Alice's request to talk to Bob. In this example Alice and Bob belongs to the same realm (domain). When Alice logs onto her workstation and requests to talk to Bob the following process occurs: [KAU95, STA99]

1. Alice logs on to her workstation and requests a ticket-granting ticket.

2. Authentication server verifies Alice's access right in the database, creates a ticket-granting ticket and session key. The results are encrypted using Alice's password and sent back to Alice.

3. Alice sends the KDC (depicted as Kerberos in Figure 1) the ticket she received, Bob's name, and an authenticator, which consists of the time of day encrypted with the session key.

4. The ticket-granting server decrypts the ticket and authenticator, verifies the request, then if the request is valid, the KDC constructs a new key, for use in communication between Bob and Alice, and

the KDC also constructs a new ticket, which consists of the newly generated key, Alice's name, and an expiration time. The new ticket is encrypted with Bob's master key. The newly encrypted ticket, Bob's name, and the session key are encrypted with Alice's master key and sent back to Alice.

5. Alice decrypts the information and sends a request to Bob. The request consists of the ticket, which Alice received that is encrypted with Bob's master key (session key and Alice's name) and an authenticator (time and session key).

6. Bob decrypts the session key and discovers Alice's name and a session key. Bob decrypts the authenticator and sees that the party to which he will communicate does indeed know the session key; he also checks the time in the authenticator to ensure it is close to the current time (a step taken to mitigate replay attacks). Bob sends messages to Alice.

For more information on Kerberos, the reader can reference [MWS2v] as a starting point. As mentioned earlier, the Kerberos protocol requires the client to present a valid and correct password or private key to the server in order to receive a ticket. Most readers are familiar with passwords, but they may not be as familiar with private keys. Private keys are often associated with a PKI, which will be discussed in Section D. Talisker does not provide any documentation stating that it supports a PKI, but it does support a component that PKIs can support as well: smart cards [OWS3n].

C. SMART CARDS

Smart cards are credit card size devices with a crypto-capable microprocessor [OWS3o]. A layer of security can be added to WinCE by using smart cards as a mechanism to store authentication information or to enable digital signing. The WinCE smart card subsystem supports CrytoAPI through smart card service providers (SCSPs), which are dlls (defined in Chapter V, Section D, subsection 6) that enable access to specific services. The subsystem provides an interface between the smart card reader hardware and the applications. Windows CE does not provide SCSPs as part of the

operating system code that is shipped to the OEMs. Smart card vendors must provide the appropriate SCSPs. Windows CE does provide the interfaces shown in Table 4.1.

Subsystem component	**File**	**Description**
Resource manager	Scard.dll	Uses the Win32 APIs to manage access to multiple readers and smart cards
Resource manager helper library	Winscard.dll	Exposes PC/SC services for using smart cards and smart card readers.
Smart card reader helper library	Smclib.lib	Provides common smart card driver support routines and additional T=0[1] and T=1 protocol support to specific drivers as needed
Sample smart card reader drivers	Pscr.dll bulltlp3.dll stcusb.dll	SwapSmart PC reader driver. Serial reader driver. Universal serial bus (USB) reader driver.

Table 4.1 Windows CE Smart Card Interfaces (From: [MWS2w])

Typically, a smart card system consists of applications, a subsystem that handles communication between smart card readers and the applications, readers (hardware not included with WinCE subsystem), and the smart card. Figure 4.1 illustrates this relationship and process flow: [MWS2w]

[1] T=x is a type of data manipulation language table, see [MWS2y].

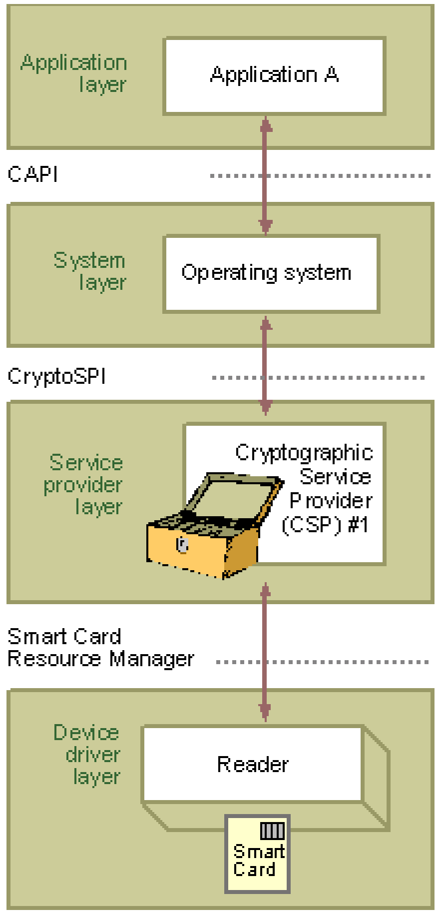

Figure 4.2 Windows CE-based Smart Card System Architecture (From: [MWS2w])

In this section, the WinCE smart card subsystem was briefly discussed but also of note is a Microsoft press release in June 2000. This press release allows the reader to understand that smart cards and smart card technology has begun its proliferation in both the business and government communities as indicated by the following quote from the article. [OWS3p]

> The General Services Administration (GSA) is an example of a forward-looking organization working to deploy smart cards to the federal government. GSA recently awarded a 10-year contract estimated at $1.5 billion to five industry partners….Microsoft is the subcontractor of each of these prime awardees.

Smart cards used as a component in the PKI architecture are used as mechanisms that store a user's private key. The smart card allows the user to access the network and/or possibly services on the network. The smart card provides a means of identifying

the user to the network and authenticating the user. The Department of Defense (DoD) plans to use smart cards as part of their PKI [DOD01].

D. DOD PUBLIC KEY INFRUSTRUCTURE (PKI)

The Department of Defense's PKI is an asymmetric cryptography key system, which means it uses two different keys to encrypt and decrypt information [DOD01]. The reader should note this key system is different from Kerberos, mentioned earlier, which uses a symmetric key system for its session key service. In PKI, the keys are complementary [DOD01]. Public Key Infrastructure has in addition to the asymmetric key other components that enable it to provide the following IA services.

- Authentication: ensures senders are who they claim to be
- Confidentiality: ensures data remains private
- Integrity: ensures data has not been modified
- Non-repudiation: ensures person sending cannot deny participation

The components that the PKI is built upon to provide the services mentioned above are as follows: [DOD01]

- **Certificate Policy**: this is a policy that establishes the common security rules for a given assurance class under which certificates are generated and managed to maintain trust within a PKI.
- **Public Key Certificates**: this is the cornerstone of the PKI technology; it is the ability to distribute public keys to large populations while maintaining the trust that each certificate is associated with the identity, public and private keys of the claimed subscriber.
- **Token**: this is a mechanism used to hold certificates and private keys.
- **Subscriber**: the customers of the products and services provided by a PKI.
- **Registration**: the process that subscribers use to identify themselves to the PKI and request certificates.

- **Certificate Management**: this is the generation, production, distribution, control, tracking, revocation, and destruction of public/private keys and associated certificates.

To understand exactly how PKI works, the reader can reference [DOD01]. This work will provide a very brief overview to give the reader a basic understanding. In a PKI a subscriber can log onto a PK-enabled computer with his or her private key, which could be stored on a smart card or 3.5” floppy disk or some other medium. The subscriber will have access to the computer and any services provided by the PKI in accordance with his or her authorizations. The subscriber’s authorization is based on his or her private key, which identifies the person.

During the system’s authentication process, the subscriber’s identity is verified along with the services that the subscriber can access. Along with providing the system with the subscriber’s private key, the subscriber should be required to enter a password or fingerprint or some other type of identifying information, to mitigate the possible loss or theft of the subscriber’s private key. This analysis will not discuss the underlying mechanisms that make a PKI work. Here it is sufficient to describe how some of the PKI components can be supported by WinCE.

As mentioned earlier, WinCE, Talisker, has a certificate database. Talisker also has various certificates preloaded as with most Microsoft browsers to support companies such as Verisign and others. Talisker supports smart cards through its CAC subsystem. The aforementioned functionality in Talisker may make it possible to modify WinCE to support PKI without overhauling the code.

E. CONCLUSION

Authentication is one facet of system security. Thus, it enhances the security of the operating system. In deciding which authentication protocol to use, developers must chose one robust enough to allow maximum participation with current and future technologies (as well as can be anticipated). Talisker provides a robust authentication protocol (Kerberos), and it also supports two key components of a DoD’s PKI: certificates and smart cards. Talisker may prove to be a bridging device between current technology and future technology.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EMERGING TECHNOLOGIES

A. MOORE'S LAW

In 1965, Gordon Moore, eventual cofounder of the Intel Corp., predicted that the density of transistors in an integrated circuit would double every year. Moore's prediction was later changed to reflect an 18-month cycle. His prediction became known as Moore's law. [OWS3e] Not only has his law proven to be accurate over the years, but it has also described the trends of microprocessor performance [OWS3e]. Today, in some areas, technology is advancing at an even faster rate than Moore's law hypothesized. For example, according to Lt. Gen. Michael Hayden, director of the National Security Agency (NSA), NSA is losing the race to keep up with technology; the development cycle of the global industry (telecommunications) is moving at the "speed of light" [OWS3f].

1. Computing at the Speed of Light (Optical or Photonic Computing)

Technology may not evolve literally at the speed of light (186,171 miles per second), but the analogy puts in perspective the pace at which technology is changing. One such change that is being researched, although real world uses may be years away, is the slowing down of light [OWS3g]. A Danish physicist, Lene Vestergaard, has developed a way to slow light down to 38 miles per hour [OWS3g]. Controlling the speed of light presents many possibilities.

> One such possibility is the development of extremely fast computers based on optical switches that would open and close under the control of very weak laser beams. Computer systems could then use optically switched logic gates instead of today's electronic logic gates. [OWS3g]

If this innovation can be realized, the power of the desktop could conceivably be in the palm of our hands.

2. Computing on the Atomic Scale (Quantum Computing)

Another processing technology that, if realized, will revolutionize computing is quantum computing. Quantum computing is based on the *quantum theory*, which arose at the beginning of the 20^{th} century to explain certain phenomena that could not be modeled using classical mechanics. Quantum computing is based on the idea of using quantum bits, called qubits, that can represent ones, zeros, or both at the same time (superposition), which is different from electronic circuits in today's computer that can represent only ones and zeros [OWS3h]. The increased number of data states in quantum computing will increase the computation power of computers, and the superposition of qubits will enable computers to apply the same operation to many numbers in parallel in one step [OWS3h]. Quantum computing would allow computer scientist to overcome computational complexity hurdles and would permit a variety of here-to-fore "impossible" computations.

B. TOMORROW (THE FUTURE)

It is difficult to estimate what will emerge, but much of the research in the areas of computing power seems to indicate that palm-sized devices will have a role in ushering in our future (whether PDAs are transitional hardware or "the future" remains to be seen). From DNA computing to chaotic computing and everything in between (photonic computing, and quantum computing), the focus is on making computers smaller, faster, and more powerful. Conceptually, computers will be a part of everything from our appliances to our transportation. In a PC Magazine Online article (*Fast-Forward to 2010*) written by Carol Levin in January 1999, she interviewed two IBM visionaries, Paul Horn – director of IBM research, and Phil Hester – chief technology officer of IBM Personal Systems Group [OWS3i]. They envision a possible future as follows:

> Your refrigerator will know when you're running low on milk and remind your hand-held device so you can pick up a quart on the way home. At the grocery store, you'll pick up a few items, toss them in a bag, and leave; radio-frequency tags on the items will charge your account automatically. Your car dealer will detect car problems remotely and correct them by downloading software repairs directly to your car over an Internet connection. A portable, miniaturized storage device will store all the

> information in the Library of Congress. They went on to say that within the next 20 years, a holographic cube will offer storage for $1 per terabyte…and digital assistants (PDAs) will give you thoughtful advice. [OWS3i]

To the observant reader, the future may appear more near than far. As mentioned in Chapter II, PDAs are becoming more popular among professionals and consumers. The Pocket PCs (Casio, IPaqs and Jornada) are continually evolving to provide improved functionality, such as improved synching capabilities, multitasking, and processing power [OWS3j]. Pocket PCs also provides new software and expansion possibilities, which can be used for everything from memory cards to modems. With smart appliances entering the market place and the proliferation of Internet connectivity for PDAs, the future may be as soon as tomorrow.

Tomorrow is our future, but today's technology is our bridge to that future. If we believe the future will be similar to that of IBM's visionaries, PDAs will have a significant role in that future. Today PDAs don't give us thoughtful advice, but they do provide scheduled reminders. Our refrigerators are not notifying our PDAs to remind us to pick up milk, but our PDAs can remind us of our shopping list. PDAs have a useful role in some people's lives today, and it is a role that is not far removed from their predicted future role. To achieve the envisioned future, there must be other technologies introduced to bridge the gap from where we are today to where we will be tomorrow. One of these is the Microsoft .NET effort.

C. TODAY (.NET)

Microsoft's ***.NET*** (pronounced "dot net") initiative has been described as an ambitious plan to build an operating system on the Internet itself [LEV01]. The .Net initiative is of interest because it may be a technology that will enable our automobile mechanics to detect problems with our cars, remotely, and notify us via email on our PDAs – a possible interim step to achieving the futuristic goal of having the mechanic repair certain problems by downloading software.

By describing .Net as an operating system on the Internet, the reader should understand that each device that connects to the operating system (Internet) is treated as a potential process – review Chapter II. (Also each device will have its own local

operating system such as Win2000[2], WinCE, WINNT, etc. The Internet operating system mentioned is for the reader to understand the illusion created by .Net applications.) Just as an operating system has to create a process, .Net has to "create a process" as well. Once a person who is a subscriber to .Net logs onto a computer connected to the Internet, a process begins on that person's behalf – consistent with the operations of an operating system. A *NewsWeek* article states,

> Microsoft will (after the .Net initiative) transform its business model to focus on subscription-based services. [LEV01]

Conceivably, through this subscription process to .Net, consumers will interact with business applications as well as other Microsoft services.

Each device that connects to the Internet and executes assemblies (a .NET component that will be explained later) is described as a potential process. As explained in Chapter II, processes reserve resources to execute programs. When a person logs onto a device, goes out to the Internet, and executes an assembly, the device becomes a process working on that user's behalf. The device reserves the resources needed for the program (assembly) to execute. In the .NET framework all the resources needed to execute an assembly may not be contained in the assembly or on the local machine [MWS2k]. Therefore proper execution of the assembly may require additional resources, which may have to be retrieved within the .NET framework. This may be transparent to the user, so, from the user's perspective, .NET may appear to be an operating system.

.NET is also described as a development platform. One article describes .NET as an open language platform for enterprise and Web development [MEY01]. Although .NET may provide numerous benefits for both users and developers, it is the developers who will notice and benefit from the ASP.NET component of .NET [MEY01]. ASP.NET maintains session state without storing client information on the server, and developers no longer have to use URL encoding or cookies [MEY01]. However, there are security implications (which will not be discussed) that developers must consider when deciding whether or not to use this capability.

---

[2] Windows 2000, Windows NT 4.0, Windows 95, Windows Me, Windows 98, Windows 98 SE and Windows CE are trademarks of the Microsoft Corporation.

There may be differences of opinion over whether or not .NET is an operating system on the Internet or strictly a development technology, but that is more a matter of perspective than functionality. Its functionality has characteristics of an operating system and of a development platform. Microsoft characterizes .NET as

> a new platform for building integrated, service-oriented applications to meet the needs of today's Internet business; applications that gather information from, and interact with, a wide variety of sources, regardless of the platforms or languages in use. [MWS2l]

Microsoft's description of .NET as "a new platform for building…" confirms that .NET is a development platform. However, in the very same description, Microsoft describes .NET's applications in terms of processes…"applications gather… a wide variety of sources." Chapter II described a process as an entity that reserved resources to enable a program to execute; in a similar sense, application must gather or reserve sources or resources to ensure proper execution. As .NET is described in more detail (detailed overview), the application's (assembly's) comparison to a process should become more obvious.

D. DETAILED OVERVIEW OF .NET

The .NET structure, to include its internals and components, is more concrete. The .NET framework is much too vast to discuss in detail in one chapter, but an overview of the technology can be adequately explained. At the heart of .NET is a desire to standardize and integrate proprietary information [MWS2l]. To fulfill the requirement of standardization, .NET has embraced the **E**xtensible **M**arkup **L**anguage (XML) standard for describing data [MWS2l].

> XML is a language for describing data elements. It describes the attributes of the data and identifies its intended meaning and use. [MUR01]

In addition to using a common standard, developers also need to use a common protocol; .NET will use **S**imple **O**bject **A**ccess **P**rotocol (SOAP) [MWS2m].

Overall, .NET has six layers of structure [MEY01]. Figure 5.1 shows the six layers. The top layer in the .NET structure is the Web Services layer. The layer below Web Services is the Framework and Libraries layer followed by the Interchange and Development Environment layer, which has two separate components: interchange

standards and development environment. The next layer down is the Component Model layer followed by the Object Model layer. The final or bottom layer is the Common Language Runtime layer [MEY01].

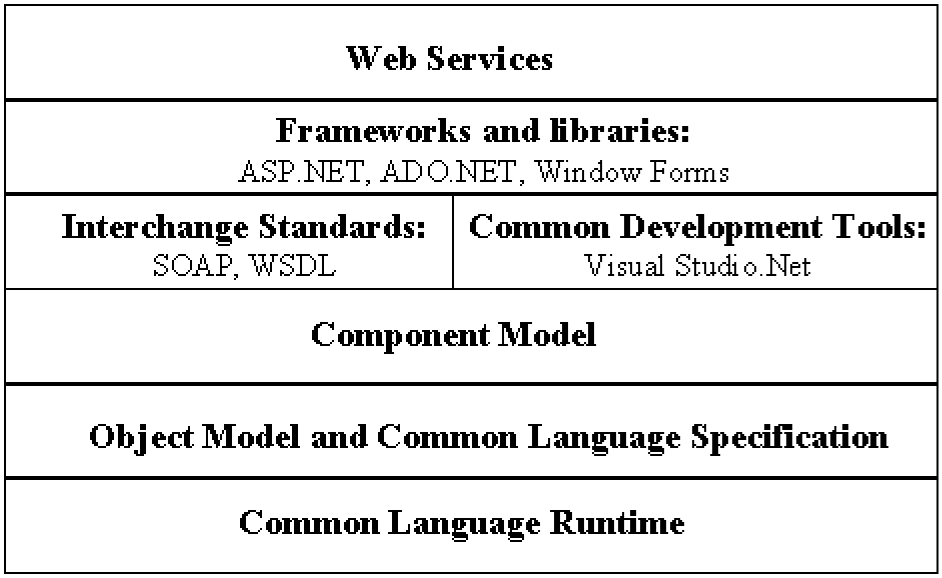

Figure 5.1 .NET Structure (From: [MEY01])

1. Web Services

Web Services is designed to provide .NET users, persons and companies, with services for e-commerce and business-to-business applications [MEY01]. Microsoft's .NET user interface promises to fulfill Berners-Lee's (inventor of the Web) vision of the Web being a collaborative (i.e., multi-user read/write) environment [MWS2m]. A Web Service is a URL–addressable resource that programmatically returns information to clients [MWS2n]. Clients can use Web Services without worrying about the implementation details of those services [MWS2n].

The web services infrastructure is designed as follows: a discovery mechanism to locate Web Services, a service description for defining how to use those services, and standard wire formats with which to communicate [MWS2o]. Figure 5.2 details the Web Services infrastructure. The role of the Web Services directories, discovery, description and wire formats are as follows [MWS2o]:

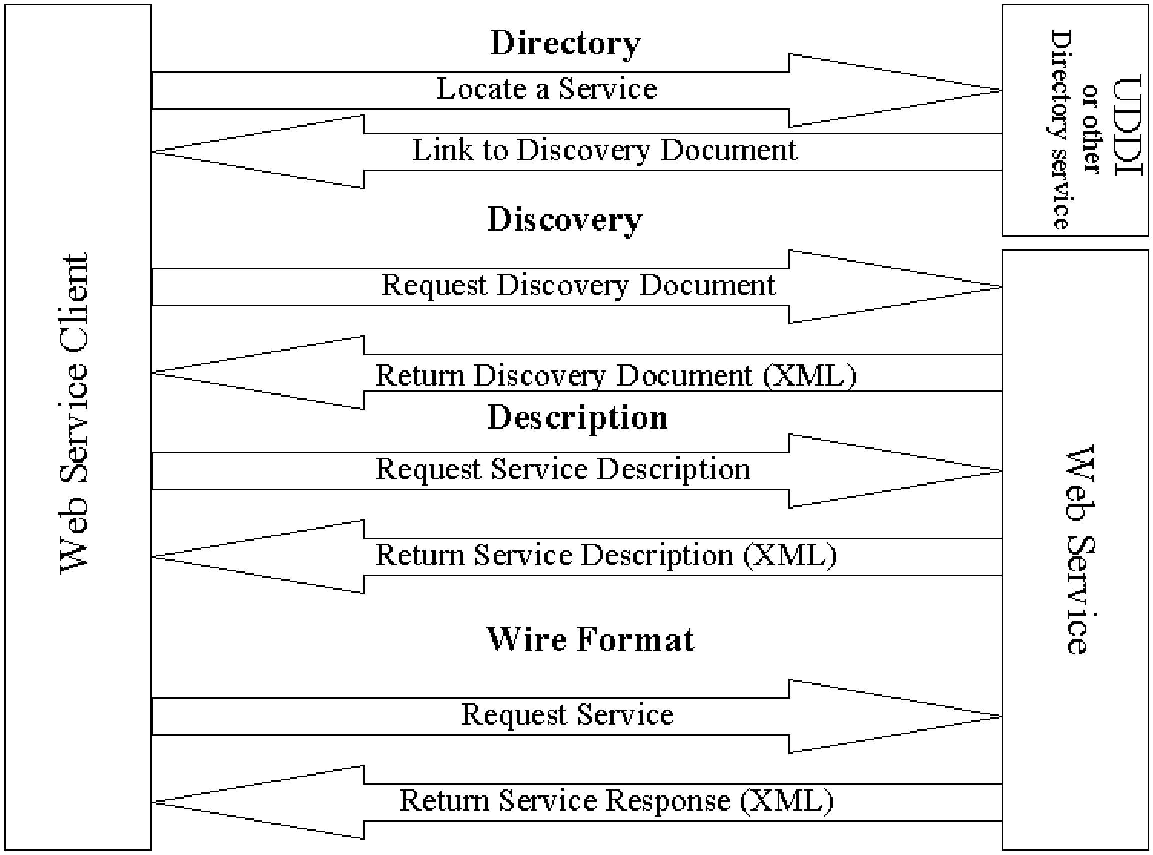

Figure 5.2 Web Service Infrastructure (After: [MWS2o])

Web Services Directories provide a central location to locate Web Services provided by other organizations. Web Services directories such as a **U**niversal **D**escription, **D**iscovery and **I**ntegration (UDDI) registry fulfills this role [MWS2o]. The XML schemas associated with UDDI define four types of information that would enable a developer to use a published Web Service: business information, service information, binding information, and information about specifications for services [MWS2p]. The *Remoting Security*[3], a security mechanism in .NET, will provide the integrity checking service; for more on the remoting security mechanism see [MWS2k].

[3] A Microsoft Corporation term – this service provides support for remote objects invocation that can span AppDomains, processes, or machines [MWS2k].

Web Services Discovery is the process of locating, or discovering one or more related documents that describe a particular Web Service using the **W**eb **S**ervice **D**escription **L**anguage (WSDL) [MWS2o].

A published .disco file, which is an XML document that contains links to other resources that describe the Web Service, enables programmatic discovery of a Web Services. [MWS2q]

A Web Service client may bypass the discovery process if the location of the service description is known [MWS2o]. When a Web Service is created for private use, there will not be a public means of finding it [MWS2q].

Web Service Clients must understand how to interact with a particular Web Service before they can use it [MWS2o]. Therefore it is necessary to provide a service description that defines what interactions the Web Service supports [MWS2o]. The service description is an XML document written in WSDL that defines the format of messages the Web Services understands [MWS2r].

Web Services Wire Formats enable universal communication by using open wire formats, which are protocols understandable by any system capable of supporting the most common Web standards. SOAP, a simple lightweight XML–based protocol, is key for providing Web Service communication. [MWS2o]

2. Frameworks and Libraries

Microsoft's .NET framework and libraries define a set of guidelines as a way to help class library designers more fully understand the trade-offs between different solutions [MWS2s]. This layer is intended to provide a well-defined managed class library with the following characteristics [MWS2s]:

Consistent: Similar design patterns are implemented across libraries.

Predictable: Functionality is easily discoverable. There is typically only one way to perform a specific task.

Web Centric: Callable from semi-trusted code.

Multilanguage: Functionality is accessible to many different programming languages.

.Net contains thousands of reusable components [MEY01]. Some of the most immediately attractive aspects for developers are ASP.NET, active server pages for developing smart Web sites and services; ADO.NET, an XML-based improvement to ActiveX Data Objects, for databases and object-relational processing; and Windows Forms for graphics [MEY01].

3. Interchange Standards/Development Environment

The third layer from the top has two different components. The interchange standard component is an XML-based standard that serves as a platform-independent means of exchanging objects [MEY01]. The most important are SOAP, an increasingly popular way to encode objects, and WSDL [MEY01]. The development environment provides a common software development environment offering facilities for development, compilation, browsing, and debugging shared by many languages [MEY01].

4. Component Model

The fourth layer from the top of the platform's structure is the component model. The component model for .NET is based on object-oriented ideas [MEY01]. In .NET, developers build *assemblies*, which consist of numerous classes with well-defined interfaces [MEY01]. Assemblies are **P**ortable **E**xecutable (PE) files [MSW2l]. Assemblies (managed PE files) are not x86 machine code or machine code targeted to any specific CPU platforms; assemblies are **M**icro**S**oft **I**ntermediate **L**anguage (MSIL) code generated by language compilers that support .NET [MSW2l]. Each assembly contains *metadata* [MSW2l].

> Metadata is used to permit communication about the data to take place between programs that do not otherwise know about each other. [MUR01]
>
> The components of an assembly are described in a manifest. [MWS2l]
>
> A manifest is a block of data that enumerates the assembly's files, and controls what types and resources are exposed outside of the assembly. The manifest also governs how references to these types and resources are mapped onto the files that contain their declarations and implementations,

> and enumerates other assemblies on which this one is dependent. The existence of a manifest provides a level of indirection between consumers of the assembly and the implementation details of the assembly and makes assemblies self-describing. [MWS2l]

For the component model layer, this paper only mentions assemblies, manifests, and metadata. The reader can find out more about the components of this layer by visiting the Microsoft development web site [MWS2x].

5. Object Model

This level provides the conceptual basis on which everything else rests, especially the object-oriented type system [MEY01].

> The formal specification of the type system implemented by the common language runtime is called the **C**ommon **T**ype **S**ystem (CTS). The CTS specifies how object classes (called types) are defined. [MWS2l]

The following is the list of possible members that a class type can contain [MWS2l]:

Field: A data variable that is part of the object's state. Fields are identified by their name and type [MWS2l].

Method: A function that performs an operation on the object, usually changing the object's state. Methods have a name, signature, and modifiers. The signature specifies the calling convention, number of parameters (and their sequence), the types of the parameters, and the type of value returned by the method. The modifiers can include custom attributes, whether the method is public, private, static, and so on [MWS2l].

Property: To the caller, this member looks like a field. But to the class implementor, this member looks like a method. Properties allow an implementor to calculate a value only when necessary and allow a class user to have simplified syntax. Properties also allow you to create read-only or write-only "fields" [MWS2l].

Event: Events provide a notification mechanism between an object and other interested object. For example, a button could offer an event that notifies other objects when the button is clicked [MWS2l].

The common language specification defines restrictions ensuring language operability [MEY01].

6. Common Language Runtime

> The common language runtime provides the basic set of mechanisms for executing .NET programs regardless of their language of origin: translation to machine code (**j**udiciously **i**ncremental **t**ranslation, or *jitting*), more commonly known as "just in time", loading security mechanisms, memory management (including garbage collection), version control, and interfacing with non-.NET code. [MEY01]

The .NET framework represents a new way of developing software by providing technologies (via the .NET common language runtime engine) that support rapid software development [MWS2l]. Some of the features provided by the .NET common language runtime engine are listed below [MWS2l]:

Consistent programming model: All application services are offered via a common object-oriented programming model, unlike today where some OS facilities are accessed via DLL functions and other facilities are accessed via COM objects.

Simplified programming model: .NET seeks to greatly simplify the plumbing and arcane constructs required by Win32 and COM. Specifically, developers no longer need to gain an understanding of the registry, GUIDs, IUnknown, AddRef, Release, HRESULTS, and so on. It is important to note that .NET doesn't just abstract these concepts away from the developer; in the new .NET platform, these concepts simply do not exist at all.

Run once, run always: All developers are familiar with "DLL Hell," but for those who are not, a dynamic link library (dll) is a collection of subroutines that can be loaded to support the execution of an application (application x). When the code in these libraries are updated (new versions), the new dll (updated version) might no longer support the

execution of application x. If a user downloads the new dll to support the execution of application y, the new dll, if it is an updated version of the old dll used to support application x, will over-write the old dll and the user will not be able to execute application x. Since installing components for a new application can overwrite components of an old application, the old app can exhibit strange behavior or stop functioning altogether. The .NET architecture now separates application components so that an app always loads the components with which it was built and tested. If the application runs after installation, then the application should always run. This marks the end of DLL Hell.

Execute on many platforms: Today, there are many different flavors of Windows: Windows 95, Windows 98, Windows 98 SE, Windows Me, Windows NT 4.0, Windows 2000 (with various service packs), **Windows CE**, and soon a 64-bit version of Windows 2000. Most of these systems run on *x*86 CPUs, but Windows CE and 64-bit Windows run on non-*x*86 CPUs. Once written and built, a managed .NET application (that consists entirely of managed code) can execute on any platform that supports the .NET common language runtime. *It is even possible that a version of the common language runtime could be built for platforms other than Windows in the future*. Users will immediately appreciate the value of this broad execution model when they need to support multiple computing hardware configurations or operating systems.

Language integration: COM allows different programming languages to interoperate with one another. .NET allows languages to be integrated with one another. For example, it is possible to create a class in C++ that derives from a class implemented in Visual Basic. The .NET platform can enable this because it defines and provides a type system common to all .NET languages. The Microsoft Common Language Specification describes what compiler implementors must do in order for their languages to integrate well with other languages. Microsoft provides several compilers that produce code targeting the .NET common language

runtime: C++ with managed extensions, C# (pronounced "C sharp"), Visual Basic (which now subsumes VBScript and Visual Basic for Applications), and JScript®. In addition, companies other than Microsoft are producing compilers for languages that also target the .NET common language runtime.

Code reuse: Using the mechanisms just described, you can create your own classes that offer services to third-party applications. This, of course, makes it extremely simple to reuse code and broadens the market for component vendors.

Automatic resource management: Programming requires skill and discipline. This is especially true when it comes to managing resources such as files, memory, screen space, network connections, database resources, and so on. One of the most common bugs occurs when an application neglects to free one of these resources, causing that application or others to perform improperly at some unpredictable time. The .NET common language runtime automatically tracks resource usage, guaranteeing that an application never leaks resources (in other words, this ensures the proper management of resources so that a resource is freed only after an application has finished using it and not before, it also ensures the resource is released and made available for other applications once the original application has finished using it). In fact, there is no way to explicitly free a resource.

Type safety: The .NET common language runtime can verify that all code is type safe. Type safety ensures that allocated objects are always accessed compatibility. Hence, if a method input parameter is declared as accepting a 4-byte value, the common language runtime will detect and trap attempts to access the parameter as an 8-byte value. Similarly, if an object occupies 10 bytes in memory, the application can't coerce this into a form that will allow more than 10 bytes to be read. Type safety also means that execution flow will only transfer to well-known locations (namely, method entry points).

Rich debugging support: Because the .NET common language runtime is used for many languages, it is now much easier to implement portions of applications using the language that is best suited for it, the application. The .NET common language runtime fully supports debugging applications that cross language boundaries. The runtime also provides built-in stack-walking facilities, making it much easier to locate bugs and errors.

Consistent error handling: One of the most aggravating aspects of programming in Windows is the inconsistent ways errors are reported. Some functions return Win32 error codes, some return HRESULTS (which is a return value of COM functions and methods [MWS2u]), and some raise exceptions. In .NET, all errors are reported via exceptions. Exceptions allow the developer to isolate the error-handling code from the code required to get the work done. This greatly simplifies writing, reading, and maintaining code. In addition, exceptions work across module and language boundaries as well.

Deployment: Today, Windows-based applications can be incredibly difficult to install and deploy. There are usually several files, registry settings, and shortcuts that need to be created. In addition, completely uninstalling an application is nearly impossible. With Windows 2000, Microsoft introduced a new installation engine that helps with all of these issues, but it is still possible that a company authoring a Microsoft Installer Package may fail to do everything correctly. .NET seeks to make these issues ancient history. .NET components are not referenced in the registry. In fact, installing most .NET-based applications will require no more than copying the files to a directory, and uninstalling an application will be as easy as deleting those files.

Security: Traditional OS security provides isolation and access control based on user accounts. This has proven to be a useful model, but at its core it assumes that all code is equally trustworthy. This assumption was justified when all code was installed from physical media (such as CD-

ROM) or trusted corporate servers. But with the increasing reliance on mobile code such as Web scripts, Internet application downloads, and e-mail attachments, there is a need for more granular control of application behavior. The .NET Code Access Security model will deliver the granular control [MWS2l]. For more on the .NET Code Access Security model see [MWS2k].

E. CONCLUSION

.NET is more complex then what has been described in this chapter. However, this chapter gives a fundlemental overview of what .Net is, what it is designed to do, and how it is designed. The focus of this chapter has been on the .NET structure. For more on .NET visit the Microsoft development web page [MWS2l]. Although .NET is being designed to work on any operating system, as of the writing of this paper, it only works on Windows-based operating systems – Windows CE being one.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSION

Pocket PC, which is the third generation of the WinCE operating system designed to run on small devices from embedded systems to Personal Digital Assistants (PDAs), is gaining popularity among professionals and is estimated to own 40% of the market share for palm-sized devices by 2004 [OWS3c]. Today, many people are using PDAs; and as PDA prices drop, additional users are anticipated in the near future. The trend among professionals towards using Pocket PC instead of Palm, which is a competing operating system, seems to indicate a preference for functionality and familiarity.

The familiarity of WinCE (for PDAs) is based on its applications, which are similar to those found on desktops executing Windows operating system. The use of WinCE for embedded devices and PDAs should raise security concerns because of recent exploits against the Palm operating system [KIN01]. Although the exploits are not against WinCE, the exploits prove that these small-foot printed operating systems are vulnerable. This vulnerability puts at risk information stored on PDAs as well as appliances running WinCE as an embedded OS.

Mitigating these risks require security. Security must be implemented to allow the operating system to continue to perform at a reasonable level (a subject for further research) while ensuring privacy of data and protection of the operating system itself to include the data, code, and mechanisms (dependencies) of the operating system. Any success in enhancing the security of WinCE requires understanding the operating system through analysis of its code. That analysis is facilitated through an understanding of the operating system itself.

This portion of the project examined WinCE's threads and processes data structures, creation, and termination. This work also looked at the authentication mechanism in *Talisker*, the next generation of the WinCE operating system. Talisker

supports Kerberos, an authentication protocol based on tickets, and session keys (symmetric keys). Talisker also supports components of a PKI, which is a key management scheme based on asymmetric keys (public keys, private keys and certificates). Public key infrastructure delivers authentication, integrity, confidentiality, and non-repudiation.

PKI does not make an operating system more self-protecting. A public key infrastructure provides a means for identifying an object (authentication), providing integrity and confidentiality of the data being transferred between objects, and ensuring that an object/subject cannot deny executing a transaction (non-repudiation). Thus, PKI is a means for providing security to data. However, authentication, which is used to identify an object/subject, does provide limited security to the operating system.

Authentication can be used to insure that only authorized individuals gain access to the PDA. By providing such a mechanism, for example, by requiring a user to provide a valid smart card with an authorized pass phrase bound to that particular smart card, and by having a valid certificate in the PDA's certificate database before the system (PDA) will initialize, provides initial protection to the device. This work also discussed future technologies that may affect WinCE. Faster processors may make it possible to enhance security without degrading performance to unacceptable levels. Personal Digital Assistants may be required to support new network technologies, such as .NET. To avoid overlooking security requirements for these future technologies, developers must decide what support to implement and how to ensure security with the added functionality.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

There are a few areas for future research. In the WinCE operating system, the file system, the I/O system, the memory and cache management, etc., should be examined. Through code analysis it should be determined whether or not WinCE can be modified to fully support PKI without having to completely rewrite the operating system. By supporting PKI, conceivably, Pocket PC could be integrated into an "unclassified but sensitive" network without degrading the network's integrity. This is theoretically based on using a combination of technologies currently supported by *Talisker*, smart cards and

the PDA's unique identification. Further research is required to prove whether or not this is possible.

In this work, there were certain security attributes in the create process function that were not used: *lpsaProcess* and *lpsaThread* (Chapter III, Section C, Subsection 1). Several areas of further research that could prove fruitful include determining what these attributes would provide to the operating system in terms of security; and if these attributes provide additional security, how they can be supported without significant modification of the code.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

A. BOOKS AND ARTICLES

[AND72] Anderson, J.P., "Computer Security Technology Planning Study.", ESD-TR-73-51, Vol. 1, Hanscom AFB, MA, DTIC-AD-758206, (1972)

[ATT76] Attanasio, C.R., Markstein, P.W., and Phillips, R.J., "Penetrating an Operating System: A Study of VM/370 integrity." *IBM System Journal No. 1* 102-116, (1976)

[BAC86] Bach, M. J., *The Design of the UNIX Operating System*, Prentice-Hall, Inc, (1986)

[BOS00] Boswell, W., *Inside Windows 2000 Server*, New Riders Publishing (2000)

[CCI99] Common Criteria for Information Technology Security Evaluation, Version 2.1, CCIMB-99-031 (Aug 1999)

[DIJ68] Dijkstra, E.W., "The Structure of the "THE"-Multiprogramming System." *ACM Symposium on Operating Systems Principles, Vol. 11, No. 5* 341-345 (May 1968)

[DOD01] Department of Defense, "PKI: Unlocking the Door to eBusiness." Department of Defense brochure (2001)

[GAR99] Gareau, J.L., *Windows CE From the Ground Up*, Annabooks, (1999)

[GOL79] Gold, B.D., Linde, R.R., Peeler, R.J., Schaefer, M., Scheid, J.F., and Ward, P.D., "A Security Retrofit of VM/370." *National computer Conference* 335-344 (1979)

[KAU95] Kaufman, C., Perlman, R., and Speciner, M., *Network Security Private Communication in a Public World*, Prentice-Hall, Inc, (1995)

[KIN01] Kingpin, and Mudge., "Security Analysis of the Palm Operating System and its Weaknesses Against Malicious Code Threads." *Proceedings of the 10th USENIX Security Symposium* (Aug 2001)

[LEV01] Levy, S., " 'A Cloud Lifted.' " *Newsweek* 38-39, (July 9, 2001)

[LIN75] Linde, R.R., "Operating System Penetration." *National Computer Conference* 361-368, (1975)

[MEY01] Meyer, B., ".NET is Coming." *Computer Innovative Technology for Computer Professionals* 92-97 (August 2001)

[MIC98] Microsoft Corporation, *Microsoft Windows CE Programmer's Guide*, Microsoft Press, (1998)

[MIC99] Microsoft Corporation, *Microsoft Windows CE Programmer's Guide*, Microsoft Press, (1999)

[MUL93] Mullender, S., *Distributed Systems*, ACM Press New York, (1993)

[MUR98] Murray, J., *Inside Microsoft Windows CE*, Microsoft Press, (1998)

[MUR01] Murray, W.H., "Security for XML and Other Meta-data Languages." (2001)

[SAL73] Saltzer, J.H., "Protection and the Control of Information Sharing in Multics." *Communications of the ACM, Vol. 17, No. 7* 388-402 (Jul 1974)

[SAL75] Saltzer, J.H., and Schroeder, M.D., "The Protection of Information in Computer Systems." *Proceedings of the IEEE, Vol.63, No. 9* 1278-1308, (Sept 1975)

[SCH75] Schroeder, M.D., "ENGINEERING A SECURITY KERNEL FOR MULTICS." *Proceedings of The Fifth Symposium on Operating Systems Principles*, 25-31 (Nov 1975)

[SCH77] Schroeder, M.D., Clark, D.D., and Saltzer, J.H., "The Multics Kernel Design Project." *Proceedings of Sixth ACM Symposium on Operating Systems Principles*, 43-56 (Nov 1977)

[SIL94] Silberschatz, A., and Galvin, P.B., *Operating System Concepts*, Addison-Wesley Publishing Company, Inc, (1994)

[SIL98] Silberschatz, A., and Galvin, P.B., *Operating System Concepts*, Fifth Edition, John Wiley & Sons, (1998)

[SOL98] Solomon, D.A., *Inside Windows NT*, Second Edition, Microsoft Press, (1998)

[SOL00] Solomon, D.A., and Russinovich, M.E., *Inside Microsoft Windows 2000*, Third Edition, Microsoft Press, (2000)

[STA98] Stallings, W., *Operating Systems Internals and Design Principles*, Third Edition, Prentice-Hall, Inc, (1998)

[STA99] Stallings, W., *Cryptography and Network Security Principles and Practice*, Third Edition, Prentice-Hall, Inc, (1999)

[TAN97] Tanenbaum, A.S., and Woodhull, A.S., *Operating Systems: Design and Implementations*, Second Edition, Prentice-Hall, Inc, (1997)

B. OTHER

1. Newsgroups and Mailing Lists [NGS1]

a. BUGTRAQ@securityfocus.com

b. microsoft.public.windowsce.talisker.techpreview

2. Microsoft MSDN Websites [MWS2]

a. Microsoft Corporation. (2001). Introducing Microsoft Windows CE 3.0. *MSDN library*.

http://msdn.microsoft.com/library/techart/introwince.htm (17 Apr 2001)

b. Yao, P. (Nov 2000). Windows CE 3.0: Enhanced real-time features provide sophisticated thread handling. *MSDN magazine*.

http://msdn.microsoft.com/library/periodic/period00/RealCE.htm (21 Jun 2001)

c. Microsoft Corporation. (2000). Microsoft Windows CE 3.0 kernel services: multiprocessing and thread handling. *MSDN library*.

http://msdn.microsoft.com/library/techart/threads30.htm (21 Jun 2001)

d. Microsoft Corporation. (2000). About processes and threads. *MSDN library*.

http://msdn.microsoft.com/library/psdk/winbase/prothred_0n03.htm (17 Apr 2001)

e. Microsoft Corporation. (2001). CreateProcess. *MSDN library*.

http://msdn.microsoft.com/library/en-us/wcesdkr/htm/_wcesdk_win32_createprocess.asp (22 Jun 2001)

f. Microsoft Corporation. (2001). Terminating a process. *MSDN library*.

http://msdn.microsoft.com/library/psdk/winbase/prothred_9583.htm (17 Apr 2001)

g. Microsoft Corporation. (2000). Processes and threads. *MSDN library*.

http://msdn.microsoft.com/library/psdk/winbase/prothred_86sz.htm

(17 Apr 2001)

h. Microsoft Corporation. (2000). Working with processes and threads in Microsoft Windows CE 2.1. *MSDN library*.

http://msdn.microsoft.com/library/techart/threads21.htm (17 Apr 2001)

i. Microsoft Corporation. (2001). CreateThread. *MSDN library*.

http://msdn.microsoft.com/library/en-us/wcesdkr/htm/_wcesdk_win32_createthread.asp (6 Jul 2001)

j. Microsoft Corporation. (2001). TerminateThread. *MSDN library*.

http://msdn.microsoft.com/library/en-us/wcesdkr/htm/_wcesdk_win32_terminatethread.asp (6 Jul 2001)

k. Richter, J. (Oct 2000). Part 2: Microsoft .NET framework delivers the platform for an integrated, service oriented web. *MSDN magazine*.

http://msdn.microsoft.com/msdnmag/issues/1000/Framework/print.asp (2 Aug 2001)

l. Richter, J. (Sep 2000). Microsoft .NET framework delivers the platform for an integrated, service oriented web. *MSDN magazine*.

http://msdn.microsoft.com/msdnmag/issues/0900/Framework/print.asp (2 Aug 2001)

m. Smith, M. (2001). Microsoft .NET: Is this Internet strategy the next big thing? *MSDN library*.

http://msdn.microsoft.com/library/en-us/dnw2kmag00/html/MicrosoftNET.asp (15 Aug 2001)

n. Srinivasan, P. (2001). An introduction to Microsoft .NET remoting framework. *MSDN library*.

http://msdn.microsoft.com/library/en-us/dndotnet/html/remoting.asp (15 Aug 2001)

o. Microsoft Corporation. (2001). Web services infrastructure. *.NET Framework Developer's Guide*.

http://msdn.microsoft.com/library/en-us/cpguidnf/html/cpconwebservicesinfrastructure.asp (17 Aug 2001)

p. Microsoft Corporation. (2001). Web services directories. *.NET Framework Developer's Guide*.

http://msdn.microsoft.com/library/en-us/cpguidnf/html/cpconwebservicesdirectories.asp (17 Aug 2001)

q. Microsoft Corporation. (2001). Web services discovery. *.NET Framework Developer's Guide*.

http://msdn.microsoft.com/library/en-us/cpguidnf/html/cpconwebservicesdiscovery.asp (17 Aug 2001)

r. Microsoft Corporation. (2001). Web services description. *.NET Framework Developer's Guide*.

http://msdn.microsoft.com/library/en-us/cpguidnf/html/cpconwebservicesdescription.asp (17 Aug 2001)

s. Microsoft Corporation. (2001). Net frameworks design and guidelines *.NET Framework Developer's Guide*.

http://msdn.microsoft.com/library/en-us/cpapndx/html/_cor_introduction_to_the_netframeworks_design_guidelines.asp (17 Aug 2001)

t. http://msdn.microsoft.com/library/en-us/dllproc/hh/winbase/prothred_0b81.asp

u. Microsoft Corporation. (2001). HRESULT definitions. *MSDN library*.

http://msdn.microsoft.com/library/en-us/com/hh/com/error_6xo3.asp (4 Sep 2001)

v. Brown, K. (2000). Understanding Kerberos Credential Delegation in Windows 2000 Using the TktView Utility. *MSDN Security Brief*.

http://msdn.microsoft.com/msdnmag/issues/0500/Security/Security0500.asp (9 Jul 2001)

w. Alforque, M. (2000). Creating a Secure Windows CE Device. *Library*.

http://msdn.microsoft.com/library/techart/winsecurity.htm (18Apr 2001)

x. http://msdn.microsoft.com

y. Microsoft Corporation. (2001). Data Manipulation Language. *Platform SDK*.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/indexsrv/hh/indexsrv/ixoledb_58dv.asp (21 Sep 2001)

3. Other Websites [OWS3]

a. http://www.howstuffworks.com

b. Thornton, C. (Apr 2001). Palm vs. Pocket PC. *PC World magazine.*
http://www.pcworld.com/features/article/0,aid,41466,pg,1,00.asp (8 Jun 2001)

c. Thornton, C. (Apr 2001). Pocket PC or Palm: Which will win out? *PC World magazine.*
http://www.pcworld.com/features/article/1,aid,41466,pg,11,00.asp (8 Jun 2001)

d. Crouch, C. (Feb 2001). Tech tips: Keep your PDA data safe. *Sci-tech.*
http://www.cnn.com/2001/TECH/ptech/02/12/PDA.security.idg/index.html (8 Jun 2001)

e. Stam, N. (1999). Moore's law will continue to drive computing. *PCMagazine*.
http://netscape.zdnet.com/pcmag/features/future/moore01.html (8 Jun 2001)

f. Verton, D. (2001). NSA struggles to keep up with pace of technology.
http://www.computerworld.com/cwi/story/0%2C1199%2CNAV47_STO58331%2COO.html (8 Jun 2001)

g. Nash, S. (1999). Light slows down. *PCMagazine*.
http://netscape.zdnet.com/pcmag/news/trends/t990226a.html (8 Jun 2001)

h. Strohmeyer, R. (2000). Beyond silicon. *PCMagazine*.
http://netscape.zdnet.com/smartbusinessmag/stories/all/0,6605,2453049,00.html (8 Jun 2001)

i. Levin, C. (1999). Fast-forward to 2010. *PCMagazine*.
http://netscape.zdnet.com/pcmag/news/trends/t990127a.html (8 Jun 2001)

j. DaMommio, A. (????). An introduction to Pocket Rockets.
http://www.palmtops.about.com/gadgets/palmtops/library/weekly/aa05152001a.htm (8 Jun 2001)

k. Microsoft Windows NT workstation and Windows NT server version 4.0.
http://www.itsec.gov.uk (23 Aug 2001)

l. Shim, R. (Jul 2001). Palm losing sales race to Compaq.
http://netscape.zdnet.com/zdnn/stories/news/0,4586,5092810,00.html (15 Aug 2001)

m. Creed, A. (Aug 2000). First Palm Pilot Trojan found in the wild.
http://www.info-sec.com/viruses/00/viruses_082900a_j.shtml (15 Aug 2001)

n. Microsoft Corporation. (2001). "Talisker" Beta 2 – What's new. *Microsoft Windows Embedded*.
http://www.microsoft.com/windows/embedded/taliskerpreview/whatsnew/ (4 Sep 2001)

o. Microsoft News. (8 Oct 1999). Microsoft holds smart card '99 business development conference to showcase new smart card technology solutions. *PressPass*.
http://www.microsoft.com/presspass/press/1999/Oct99/SmartCardPR.asp (4 Sep 2001)

p. Microsoft News. (29 Jun 2000). Microsoft President and CEO Steve Ballmer outlines role for smart cards in Microsoft .NET generation software. *PressPass*.
http://www.microsoft.com/presspass/press/2000/Jun00/SmartCardDayPR.asp (4 Sep 2001)

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, Virginia 22134-5027

5. Director, Marine Corps Research Center
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, Virginia 22134-5107

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

7. Carl Siel
   Space and Naval Warfare Systems Command
   PMW 161
   Building OT-1, Room 1024
   4301 Pacific Highway
   San Diego, CA 92110-3127

8. Commander, Naval Security Group Command
Naval Security Group Headquarters
9800 Savage Road
Suite 6585
Fort Meade, MD 20755-6585
San Diego, CA 92110-3127

9. Ms. Deborah M. Cooper
Deborah M. Cooper Company
P.O. Box 17753
Arlington, VA 22216

10. Ms. Louise Davidson
N643
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

11. Mr. William Dawson
Community CIO Office
Washington, DC 20505

12. Ms. Deborah Phillips
Community Management Staff
Community CIO Office
Washington, DC 20505

13. Capt. James Newman
N64
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

14. Major Dan Morris
HQMC
C4IA Branch
TO: Navy Annex
Washington, DC 20380

15. Mr. Richard Hale
Defense Information Systems Agency, Suite 400
5600 Columbia Pike
Falls Church, VA 22041-3230

16. Ms. Barbara Flemming
Defense Information Systems Agency, Suite 400
5600 Columbia Pike
Falls Church, VA 22041-3230

17. Mr. Michael Green, Director
Public Key Infrastructure Program management Office
National Security Agency
9800 Savage Road
Ft. Meade, Maryland 20775

18. Dr. Cynthia E. Irvine
Computer Science Department
Code CS/IC
Naval Postgraduate School
Monterey, CA 93943

19. Mr. Daniel Warren
Computer Science Department
Code CS/Wd
Naval Postgraduate School
Monterey, CA 93943

20. Titus R. Burns
4725 Aliso Way
Oceanside, CA 92057